电气信息工程丛书

CAN总线技术与应用系统设计

龙志强　李晓龙　窦峰山　郝阿明　编著





机械工业出版社
CHINA MACHINE PRESS

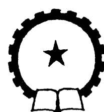
VISIT…

LANZAROTE Caliente.COM

电气信息工程丛书

CAN 总线技术与应用系统设计

龙志强　李晓龙　窦峰山　郝阿明　编著



机 械 工 业 出 版 社

本书根据作者多年来从事CAN总线教学和科研实践经验编写而成。在介绍CAN总线基本概念、技术规范基础上，介绍了CAN总线控制器SJA1000和典型CAN总线驱动器的应用，详细介绍了3种典型的具有CAN总线接口的微处理器及应用，重点对CAN总线与计算机的接口进行了分析与设计，论述了CAN总线的工程应用，给出了CAN总线的应用层协议，最后介绍了CAN总线的工程应用案例。书中所给出的相关原理图和示例程序可供读者应用时参考，这些资料已通过了实践验证。每章配有习题，以指导读者进行深入的学习。

本书不仅可供有关工程技术人员参考，也可作为自动化专业高年级本科生、相关专业控制类研究生的教材。

图书在版编目（CIP）数据

CAN 总线技术与应用系统设计 / 龙志强等编著. —北京：机械工业出版社，2013.4
（电气信息工程丛书）
ISBN 978-7-111-41867-2

Ⅰ. ①C… Ⅱ. ①龙… Ⅲ. ①总线—系统设计 Ⅳ. ①TP336

中国版本图书馆CIP数据核字（2013）第053728号

机械工业出版社（北京市百万庄大街22号 邮政编码100037）
策划编辑：时 静
责任编辑：时 静 崔利平
责任印制：杨 曦

北京诚信伟业印刷有限公司印刷

2013年4月第1版·第1次印刷
184mm×260mm·17印张·418千字
0001－3500册
标准书号：ISBN 978-7-111-41867-2
定价：39.90元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务	网络服务
社服务中心：（010）88361066	教 材 网：http://www.cmpedu.com
销 售 一 部：（010）68326294	机工官网：http://www.cmpbook.com
销 售 二 部：（010）88379649	机工官博：http://weibo.com/cmp1952
读者购书热线：（010）88379203	封面无防伪标均为盗版

前　　言

CAN 是控制器局域网络（Controller Area Network）的简称，它属于现场总线的范畴，是一种能有效支持分布式控制或实时控制的串行通信网络，并于 1993 年成为国际标准（ISO 11898）。由于 CAN 总线的良好性能和独特设计，越来越受到人们的重视，尤其在汽车领域上的应用最广泛，世界上一些著名的汽车制造厂商，大都采用了 CAN 总线来实现汽车内部的控制系统、检测和执行机构间的数据通信。同时，由于 CAN 总线本身的特点，其应用范围目前已不再局限于汽车行业，而向企业自动化、航空、航天、航海、机械工业、纺织机械、农用机械、工程机械、机器人、数控机床、医疗器械以及军事装备等领域发展，并已被公认为几种最有前途的现场总线之一。

本书根据作者 10 多年来从事 CAN 总线教学和科研实践经验编写而成，书中 CAN 总线基础的部分内容（主要为第 2、3 章）在作者编写的《现场总线控制网络技术》（机械工业出版社出版）中已经有部分介绍，考虑书稿内容的完整，本书引用并扩充了这些部分的内容。作者从 1999 年开始接触并应用 CAN 总线，并将 CAN 总线及其网络控制技术引入本科生和研究生的课堂教学或实践教学中，由于当时 CAN 总线方面教材缺乏，着手编写了 CAN 总线的内部讲义，并使用了近 10 年；目前在 CAN 总线领域已经出版了 20 多本各种层次、各具特色的专著或教材，另外周立功公司在其网站上也公布了他们整理或翻译的许多 CAN 总线资料，这些内部讲义和资料为本书的编写提供了参考。在作者的科研工作中长期应用 CAN 总线，也开发了多种 CAN 总线的计算机接口卡，这些接口卡不仅在自己课题组使用，也在全校范围内进行了推广共享，积累了有益的经验，也有了一些新的研发体会。因此，本书在编写过程中，不仅强调由浅入深，逐步介绍 CAN 总线的基本概念和理论，更是着眼系统讲技术，通过典型系统或案例的分析，让读者能够尽快熟悉并达到会应用的程度，书中的原理图和示例程序读者可以进行参考，希望这些开发实例能对读者有所帮助。

本书研究内容和典型案例主要由本研究所老师和学生的研究成果整理而成，但也参考了许多前人成果，在参考文献中也力求详细给出，可能不一定全面，在这里对有关作者表示歉意。CAN 总线技术发展迅速且应用越来越广泛，由于某些新技术、新应用作者未作深入研究，也就不便录入。希望书中涉及的一些共性技术和设计方法能对读者起到抛砖引玉的作用。

本书由龙志强主编并统稿；第 6 章、第 8 章和 1.2 节由李晓龙负责编写；5.1 节和 5.2 节由窦峰山负责编写；5.3 节和 6.6 节由郝阿明负责编写；其余章节由龙志强编写。为了尽可能给读者呈现更多的 CAN 总线在工业控制领域的应用实例，龚云生同志为本书的第 9.5 节提供

了 CAN 总线在工业控制领域的应用实例。本书在完稿过程中得到了作者研究生们的大力支持，除本书的作者外，陈杨、吴军、薛松、戴春辉、何宁、宋香磊、张大鹏、冯奕陆、谭庆龙等先后参加书中的案例实验和科研项目的研发工作，在此致以衷心的感谢。还要特别感谢机械工业出版社，正是他们的大力支持才使本书顺利出版。

由于作者水平有限，书中难免存在不妥之处，请读者见谅，并提出宝贵意见。

龙志强

目　　录

前言
第 1 章　绪论 …… 1
1.1　概述 …… 1
1.1.1　现场总线概念 …… 1
1.1.2　现场总线的产生 …… 1
1.1.3　现场总线的技术特点 …… 2
1.1.4　现场总线的技术标准 …… 3
1.2　现场总线技术基础 …… 4
1.2.1　数据通信技术基础 …… 4
1.2.2　网络拓扑 …… 12
1.2.3　网络的传输介质 …… 14
1.2.4　网络传输的介质访问控制方式 …… 17
1.2.5　现场总线通信模型 …… 18
1.3　CAN 总线基础知识 …… 19
1.3.1　CAN 总线的发展历程 …… 19
1.3.2　CAN 总线的通信方式 …… 20
1.3.3　CAN 总线的技术特点 …… 21
1.4　本章小结 …… 22
思考题与习题 …… 22
第 2 章　CAN 总线技术及其协议规范 …… 23
2.1　CAN 总线技术及其协议规范概述 …… 23
2.2　CAN 总线的系统构成 …… 23
2.2.1　CAN 总线的系统组成 …… 24
2.2.2　CAN 总线的拓扑结构 …… 24
2.2.3　CAN 总线的传输介质 …… 26
2.3　CAN 总线通信参考模型 …… 26
2.4　CAN 总线报文的传送 …… 27
2.5　CAN 总线报文的帧结构 …… 28
2.6　CAN 总线报文的编码、滤波和校验 …… 33
2.7　CAN 总线报文的优先级确定问题 …… 34
2.7.1　CAN 总线的仲裁过程 …… 34
2.7.2　数据帧和远程帧的优先级 …… 34
2.7.3　标准格式和扩展格式的优先级 …… 34

2.8 CAN 总线错误处理 35
2.9 CAN 总线故障界定 35
2.9.1 故障界定方法 36
2.9.2 错误计数规则 36
2.10 CAN 总线的位定时 37
2.11 CAN 总线的位同步 38
2.12 本章小结 39
思考题与习题 39
第 3 章 CAN 总线控制器 SJA1000 及其应用 41
3.1 SJA1000 概述 41
3.2 SJA1000 的内部结构及其控制模块 43
3.3 SJA1000 基本模式下的寄存器 44
3.3.1 基本模式下的寄存器 44
3.3.2 基本模式下的控制寄存器 45
3.3.3 基本模式下的数据段寄存器 47
3.4 SJA1000 扩展模式下的寄存器 48
3.4.1 扩展模式下的寄存器 48
3.4.2 扩展模式下的控制寄存器 54
3.4.3 扩展模式下的数据段寄存器 62
3.5 两种模式的公共寄存器 66
3.6 SJA1000 的读写时序分析 71
3.7 基于 51 系列单片机的 CAN 智能节点设计 72
3.7.1 硬件设计 72
3.7.2 软件设计 73
3.8 本章小结 75
思考题与习题 75
第 4 章 典型 CAN 总线驱动器 76
4.1 CAN 总线驱动器概述 76
4.2 CAN 总线驱动器 PCA82C250/251 76
4.2.1 PCA82C250/251 的主要特性 77
4.2.2 PCA82C250/251 的基本性能 77
4.2.3 PCA82C250/251 的功能描述 79
4.2.4 PCA82C250/251 的典型应用 80
4.3 高速 CAN 总线驱动器 TJA1040 82
4.3.1 TJA1040 的主要特性 82
4.3.2 TJA1040 的基本性能 82
4.3.3 TJA1040 的功能描述 84
4.3.4 TJA1040 的典型应用 85
4.4 高速 CAN 总线驱动器 TJA1050 86

4.4.1 TJA1050 的主要特性 86
4.4.2 TJA1050 的基本性能 86
4.4.3 TJA1050 的功能描述 87
4.4.4 TJA1050 的典型应用 88
4.5 几种典型的 CAN 总线驱动器的比较 89
4.5.1 应用方面的区别 90
4.5.2 引脚的区别 90
4.5.3 工作的模式区别 91
4.6 本章小结 92
思考题与习题 93
第 5 章 具有 CAN 总线接口的微处理器及应用 94
5.1 C8051F 系列单片机的 CAN 接口及其应用 94
5.1.1 C8051F50X 系列单片机介绍 94
5.1.2 C8051F50X 内部 CAN 控制器介绍 95
5.1.3 C8051F50X 内部 CAN 寄存器介绍 98
5.1.4 基于 C8051F500 的 CAN 硬件设计 100
5.1.5 基于 C8051F500 的 CAN 软件设计 101
5.2 TMS320F28335 DSP 的 CAN 接口及其应用 105
5.2.1 TMS320F28335 介绍 105
5.2.2 TMS320F28335 内部 eCAN 控制器介绍 106
5.2.3 TMS320F28335 内部 eCAN 寄存器介绍 108
5.2.4 基于 TMS320F28335 的 CAN 硬件设计 113
5.2.5 基于 TMS320F28335 的 CAN 软件设计 114
5.3 基于 ARM® Cortex™-M3 内核的 STM32F107 微控制器 CAN 接口及其应用 118
5.3.1 STM32F107 芯片介绍 118
5.3.2 STM32F107 的 CAN 控制器概述 118
5.3.3 STM32F107 的 CAN 控制器操作 120
5.3.4 基于 STM32F107 的 CAN 硬件设计 122
5.3.5 基于 STM32F107 的 CAN 软件设计 123
5.4 本章小结 125
思考题与习题 125
第 6 章 CAN 总线与计算机的接口设计 126
6.1 PC-104 总线 CAN 接口卡设计 126
6.1.1 PC-104 总线介绍 126
6.1.2 硬件电路设计说明 128
6.1.3 PC-104 接口卡软件设计 132
6.2 ISA 总线 CAN 接口卡设计 142
6.2.1 ISA 总线简介 142

6.2.2 硬件电路设计说明……143
6.2.3 ISA 接口卡软件设计……148
6.3 PCI 总线 CAN 接口卡设计……153
6.3.1 PCI 总线简介……153
6.3.2 硬件电路设计说明……154
6.3.3 PCI 接口卡软件设计……155
6.4 PC 并行端口与 CAN 接口设计……166
6.4.1 PC 并行端口简介……166
6.4.2 基于 EPP 模式的接口电路设计……169
6.4.3 并口接口卡软件设计……169
6.5 USB 总线与 CAN 接口设计……180
6.5.1 USB 总线简介……180
6.5.2 硬件电路设计说明……180
6.5.3 USB 接口卡软件设计……181
6.6 以太网与 CAN 接口设计……191
6.6.1 以太网简介……191
6.6.2 硬件电路设计说明……192
6.6.3 以太网接口卡软件设计……194
6.7 本章小结……198
思考题与习题……198
第 7 章 CAN 总线的工程应用问题……199
7.1 CAN 总线的扩展模式功能分析……199
7.1.1 寄存器和 RAM 地址分配……199
7.1.2 错误处理功能……200
7.1.3 仲裁丢失捕捉功能……203
7.1.4 单次发送功能……203
7.1.5 自动位速率检测功能……204
7.1.6 仅听模式……204
7.1.7 CAN 的自测试……204
7.2 CAN 总线的滤波器配置问题……205
7.2.1 BasicCAN 模式的验收滤波……205
7.2.2 PeliCAN 模式的验收滤波……205
7.3 CAN 总线的驱动问题……207
7.3.1 CAN 接口的斜率控制功能……207
7.3.2 最大总线线路长度及节点数……209
7.3.3 总线终端和拓扑结构……214
7.4 CAN 总线的实时性问题分析……219
7.4.1 CAN 总线系统的实时性问题……219
7.4.2 CAN 总线延时分析……219

7.4.3 CAN 总线延时变化分析 223
7.4.4 实时性能提升策略 224
7.5 本章小结 225
思考题与习题 225
第 8 章 CAN 总线的应用层协议 226
8.1 CAN 总线的应用层协议概述 226
8.2 CANopen 技术协议 226
8.2.1 对象字典 227
8.2.2 CANopen 子协议 228
8.2.3 CANopen 数据传输机制 228
8.3 DeviceNet 协议 229
8.3.1 DeviceNet 概述 229
8.3.2 DeviceNet 对象模型 230
8.3.3 DeviceNet 的连接方案 232
8.3.4 DeviceNet 报文协议 233
8.4 CAN 总线其他高层协议 234
8.4.1 CANaerospace 协议 234
8.4.2 CANKingdom 协议 234
8.4.3 SDS 协议 235
8.5 CAN 总线应用层协议对比研究 235
8.5.1 信息标识符分配系统 235
8.5.2 过程数据交换的特点 235
8.5.3 点对点通信信道 236
8.6 本章小结 236
思考题与习题 236
第 9 章 CAN 总线的工程应用案例 237
9.1 CAN 总线在汽车电子系统中的应用 237
9.1.1 汽车 CAN 总线技术方案 237
9.1.2 基于 CAN 总线的汽车远程故障诊断 238
9.2 CAN 总线在工程机械中的应用 242
9.2.1 CAN 总线在汽车起重机控制系统中的应用 242
9.2.2 CAN 总线在混凝土摊铺机控制系统中的应用 243
9.3 CAN 总线在轨道交通系统中的应用 244
9.3.1 CAN 总线在轨道交通运行控制系统中的应用 244
9.3.2 CAN 总线在内燃、电力机车的运行监控记录装置中的应用 246
9.3.3 CAN 总线在轨道交通车辆制动控制系统中的应用 247
9.4 CAN 总线在磁浮列车状态监测与故障诊断中的应用 248
9.4.1 CAN 总线在中低速磁浮列车状态监测与诊断系统的应用 248
9.4.2 CAN 总线在高速磁浮列车车载诊断系统的应用 250

9.5 CAN 总线在工业控制中的典型应用……251
9.5.1 CAN 总线在香烟包装机、卷接机生产线上的应用……251
9.5.2 CAN 总线在隧道窑控制系统中的应用……253
9.5.3 CAN 总线在网带窑控制系统中的应用……254
9.5.4 CAN 总线在双变频卷染机控制系统中的应用……255
9.5.5 CAN 总线在圆网印花机控制系统中的应用……256
9.6 本章小结……257
思考题与习题……257
参考文献……258

第 1 章　绪　　论

现场总线是应用在工业、农业以及军事装备等现场，在各种测控设备之间实现双向串行多节点数字通信的系统，它把网络化、信息化的概念引入到控制领域中，不仅可以实现高度灵活、高可靠性的分散控制，而且可以实现整系统、全企业、甚至世界范围内的信息共享。现场总线的出现大大提高了工作效率，必将对社会生产力的发展起到巨大的促进作用。

本章要点

- 现场总线的概念、产生、技术特点和标准形成；
- 现场总线技术基础；
- CAN 总线的发展历程；
- CAN 总线的技术特点。

1.1　概述

1.1.1　现场总线概念

根据国际电工委员会 IEC61158 标准的定义，现场总线（Fieldbus）是指将现场设备（如数字传感器、变送器、仪表与执行机构等）与工业控制单元、现场操作站等互连而成的通信网络，其关键标志是能支持双向、分散、多节点、总线式的全数字通信。现场总线技术使现场级设备的信息作为整个企业信息网的基础，使企业信息的采集控制直接延伸到生产现场。

因此，使用现场总线技术不但提高了通信能力和系统运行的可靠性，而且节省了系统安装时的布线费用和硬件费用，并使系统的管理和维护更加容易，代表了自动化技术的发展方向。

由于控制系统特别强调可靠性和实时性，所以，现场总线不同于一般电信网的通信，也不同于信息技术中一般计算机网络的通信。现场总线的数据通信是以引发物质或能量的运动为最终目的，其主要要求如下：

① 允许对实时响应的事件进行驱动通信；
② 具有很高的数据完整性；
③ 在电磁干扰和有地电位差的环境下能正常工作；
④ 使用专用的通信网等。

1.1.2　现场总线的产生

将计算机应用于控制系统一直是控制学者非常关注的问题，由于最初计算机价格昂贵，计算机控制系统主要采用集中式控制结构，集中控制以一台控制器为中心，通过扩展

的 I/O 接口实现各个传感器、执行器之间的通信，最早采用的是个人计算机或通用的工业控制计算机。进入 20 世纪 80 年代后，计算机技术飞速发展，价位迅速下降，使计算机在控制系统中的应用迅速普及。另外，由于控制系统日益复杂，也迫使控制系统的结构向着理想的全分布式系统迅速靠近，出现了集散控制系统（Distributed Control System，DCS），它最早出现于 20 世纪 70 年代后期，在 20 世纪 80 年代以后占据主导地位。其核心思想是集中管理、分散控制，即管理与控制相分离，上位机（又称工程师站和操作员站）用于集中监视管理，而把若干台下位机（又称现场控制站）下放到现场实现分布式控制，上、下位机之间用控制网络互连以实现相互之间的信息传递。因此，这种分布式的控制系统克服了集中控制对控制器处理能力和可靠性要求高的缺陷，但是，它也有自身难以克服的缺点。首先，不同的 DCS 制造商为达到垄断经营的目的而对其控制通信网络采用各自专用的封闭形式，不同制造商的 DCS 之间难以实现网络互连和信息共享。其次，DCS 的控制站仍然是集中的，现场信号的检测、传输和控制还是采用 4～20mA 的模拟信号，并没有彻底做到分散控制。因此，当时的 DCS 是一种封闭专用的、不具有互操作性的、不彻底的分散式控制系统，在这种情况下，用户开始对控制系统提出了提高开放性、降低成本的要求，这也就促使了现场总线的诞生。

现场总线是 20 世纪 80 年代末、90 年代初在国际上发展形成的，用于过程自动化、制造自动化、楼宇自动化等领域的现场智能设备互连互通。它作为工厂数字通信网络的基础，沟通了生产过程现场及控制设备之间及其与更高控制管理层之间的联系。它不仅是一个基层网络，而且还是一种开放式、新型全分布控制系统。这项以控制、计算机、数字通信等技术为主要内容的综合技术，已经引起世界范围的关注，成为自动化技术发展的热点，引发了自动化系统结构与设备的深刻变革。

1.1.3 现场总线的技术特点

1．现场总线的特点

现场总线是 3C 技术（计算机、通信、控制）的融合，其技术特点为信号输出全数字、控制功能全分散、标准统一全开放。具体特点如下：

（1）系统的开放性

系统开放性是指通信协议公开，各不同厂家的设备之间可实现互连与信息交换，现场总线的提出就是要致力于建立统一的工厂底层网络的开放系统。这里所指的开放是指对相关标准的一致性、公开性，强调对标准的共识与遵从。一个开放系统，它可以与任何遵守相同标准的其他设备或系统相连。

（2）互操作性与互用性

互操作性是指实现互连设备间、系统间的信息传送与沟通，可实行点对点、一点对多点的数字通信。互用性则意味着不同生产厂家的性能类似的设备可进行互换而实现互用。

（3）现场设备的智能化与功能自治性

它将传感测量、补偿计算、工程量处理与控制等功能分散到现场设备中完成，仅靠现场设备即可完成自动控制的基本功能，并可随时诊断设备的运行状态。

（4）系统结构的高度分散性

由于现场设备本身已可完成自动控制的基本功能，使得现场总线构成一种新的全分布式

控制系统的体系结构。从根本上改变了现有 DCS 集中与分散相结合的集散控制系统体系，简化了系统结构，提高了可靠性。

（5）对现场环境的适应性

现场总线工作在现场设备前端，作为工厂网络底层的现场总线，是专为在现场环境工作而设计的，它可支持双绞线、同轴电缆、光缆、射频、红外线、电力线等，具有较强的抗干扰能力，能采用两线制实现送电与通信，并可满足本质安全防爆要求等。

2．现场总线的优点

现场总线的以上特点，特别是现场总线系统结构的简化，使控制系统的设计、安装、正常生产运行及其检修维护，都体现出优越性。

（1）节省硬件数量与投资

由于现场总线系统中分散在设备前端的智能设备能直接执行多种传感、控制、报警和计算功能，所以可减少变送器的数量，不再需要单独的控制器、计算单元等，也不再需要 DCS 的信号调理、转换、隔离等功能单元及其复杂接线，从而节省了一大笔硬件投资。

（2）节省安装费用

现场总线系统的接线十分简单，由于一对双绞线或一条电缆上通常可挂接多个设备，所以电缆、端子、槽盒、桥架的用量大大减少，连线设计与接头校对的工作量也大大减少。当需要增加现场控制设备时，无需增设新的电缆，可就近连接在原有的电缆上，既节省了投资，也减少了设计、安装的工作量。据有关典型试验工程的测算资料表明，可节约安装费用 60%以上。

（3）节省维护开销

由于现场控制设备的自诊断等信息可通过数字通信送往控制室，用户可查询所有设备的运行、诊断维护信息，以便早期分析故障原因并快速排除，缩短了维护停工时间。同时由于系统结构简化、连线简单从而减少了维护工作量。

（4）用户具有高度的系统集成主动权

用户可以自由选择不同厂商所提供的设备来集成系统。避免因选择了某一品牌的产品被“框死”了设备的选择范围，不会为系统集成中不兼容的协议、接口而难倒，使系统集成过程中的主动权完全掌握在用户手中。

（5）提高系统的准确性与可靠性

由于现场总线设备的智能化、数字化，与模拟信号相比，它从根本上提高了测量与控制的准确度，减少了传送误差。同时，由于系统的结构简化，设备与连线减少，现场仪表内部功能加强，减少了信号的往返传输，提高了系统的工作可靠性。此外，由于它的设备标准化和功能模块化，所以还具有设计简单、易于重构等优点。

1.1.4 现场总线的技术标准

现场总线技术起源于欧洲，目前以欧美地区最为发达。由于这是一项带有革命性的、引领今后各领域自动化潮流的技术，各国、各公司都投入了大量的人力、财力在市场上展开了激烈的竞争。据不完全统计，世界上已出现过的总线种类近 200 种，经过多年的竞争和完善，目前较有生命力的有 10 多种，并仍处于激烈的市场竞争之中。

IEC/TC65 负责测量和控制系统数据通信国际标准化工作的 SC65C/WG6 于 1984 年就开

始着手制定现场总线标准。从技术需要与方便应用的角度来说，作为数据通信与控制网络的技术标准，必须实行单一标准。但由于种种经济、社会与技术原因，在历经 10 多年的斗争与调解努力之后，最终，经 IEC 的现场总线标准化组织投票，在 1999 年底通过 FF H1、ControlNet、Profibus、InterBus、P-NET、WorldFIP、Swift Net、FF 之高速 Ethernet（HSE）这 8 种现场总线成为 IEC61158 现场总线标准。

为了进一步完善 IEC61158 标准，IEC/SC65C 成立了 MT9 现场总线修订小组，继续这方面的工作。MT9 工作组在原来 8 种类型现场总线的基础上不断完善扩充，于 2001 年 8 月制定出由 10 种类型现场总线组成的第 3 版现场总线标准，该标准于 2003 年 4 月成为正式国际标准。

而本书所论述的 CAN 总线是德国 Bosch 公司于 1983 年为汽车应用而开发的一种能有效支持分布式控制和实时控制的串行通信网络，也属于现场总线（Fieldbus）的范畴。1993 年 11 月，ISO 正式颁布了控制器局域网 CAN 国际标准（ISO 11898）。

在现场总线的发展和标准制定中有一些现象很值得注意：

1）每种总线都有其产生的背景和应用领域。如 FF 总线主要适用于过程自动化，Profibus 较适用于制造业自动化，CAN 适用于汽车工业，Lon 适用于楼宇自动化等。

2）每种总线都力图拓展其应用领域，以扩张其势力范围。如 Profibus 在 DP 的基础上又开发出 PA，以适用于过程工业。

3）大多数总线都成立了相应的国际组织，如 WorldFIP 国际用户组织、FF 基金会、Profibus 国际用户组织、P-NET 国际用户组织、ControlNet 国际用户组织等。

4）每种总线都有一个或几个大型公司为其支撑，如 Profibus 以 Siemens 公司为主要支持，ControlNet 以 Rockwell 公司为主要背景，WorldFIP 以 ALSTOM 公司为主要后台。

5）大多数设备制造商都积极参加不止一个总线组织，有些公司甚至参加 2～4 个总线组织。

6）每种总线大多将自己作为国家或地区标准，以加强自己的竞争地位。现在的情况是：P-NET 已成为丹麦标准，Profibus 已成为德国标准，WorldFIP 已成为法国标准。这 3 种总线于 1994 年成为并列的欧洲标准 EN50170，其他总线也都形成了各组织的技术规范。

7）作为开放系统的数据通信技术，现场总线仍然坚持采用标准化道路。成为 IEC 现场总线标准子集的上述总线，成为 IEC TC178 国际标准的 DeviceNet、ASI、SDS（Smart Distributed System），以及成为 ISO 11898 标准的 CAN，都是在不同应用领域显示了各自技术优势的总线品种。它们是总线竞争中的佼佼者，属于总线中的优选对象。

1.2　现场总线技术基础

1.2.1　数据通信技术基础

1．通信系统的性能指标

通信系统的任务是传递信息，因此，信息传输的有效性和可靠性是通信系统最主要的指标。有效性是指传输信息的内容有多少，而可靠性是指接收信息的可靠程度。通信有效性实际上反映了通信系统资源的利用率，通信过程中用于传输有用报文的时间比例越高越有效。同样，真正要传输的数据信息位在所传输有用报文的时间比例越高有效性越好。

（1）有效性指标

1）数据传输速率：数据传输速率是单位时间内传送的数据量。它是衡量数字通信系统有效性的指标之一。当信道一定时，信息传输的速率越高，有效性越好。

例如，对串行传输而言，如果某一个脉冲只包含两种状态，传输速率（bit/s）由下式求得：

$$S_b = \frac{1}{T} \tag{1-1}$$

式中，T 为发送一位代码需要的最小单位时间。工业数据通信中常用的标准数据信号速率为9600bit/s、31.25kbit/s、500kbit/s、1Mbit/s、2.5Mbit/s、10Mbit/s 以及 100Mbit/s 等。

比特率的概念，比特（位，bit）是数据信号的最小单位，通信系统中的字符或字节一般由多个二进制位即多个比特来表示，一个字节往往是 8bit 或 16bit，每秒传输数据的二进制位数定义为比特率，记作 bit/s。

波特率的概念，波特是指信号大小方向变化的一个波形，把每秒传输信号的个数，即每秒传输信号波形的变化次数定义为波特率，单位为波特（baud）。比特率和波特率较易混淆，但它们是有区别的。每个信号波形可以包含一个或多个二进制位。若单比特信号的传输速率为 9600bit/s，则其波特率为 9600baud，它意味着每秒可传输 9600 个二进制脉冲。如果信号波形由 2 个二进制位组成，当传输速率为 9600bit/s 时，则其波特率只有 4800baud。

在讨论信道特性，特别是传输频带宽度时，通常采用波特率；在涉及系统实际的数据传送能力时，则使用比特率。

2）频率利用率：频率利用率是指单位频带内的传输速度。它是衡量数据传输系统有效性的重要指标。单位为$(\mathrm{bit/s})\cdot\mathrm{Hz}^{-1}$，即每赫兹所能实现的比特率。由于传输系统的带宽通常不同，所以通信系统的有效性仅仅看比特率是不够的，还要看其占用带宽的大小。

3）协议效率：协议效率是衡量通信系统软件有效性的指标之一。协议效率是指所传输的数据包中的有效数据位与整个数据包长度的比值。一般是用百分比表示，它是对通信帧中附加量的量度。不同的通信协议通常具有不同的协议效率。协议效率越高，其通信有效性越好。在通信参考模型的每个分层，都会有相应的层管理和协议控制的加码。从提高协议编码效率的角度来看，减少层次可以提高编码效率。

4）通信效率：通信效率定义为数据帧的传输时间同用于发送报文的所有时间之比。其中数据帧的传输时间取决于数据帧的长度、传输的比特率以及要传输数据的两个节点之间的距离。这里用于发送报文的所有时间包括竞用总线或等待令牌的排队时间、数据帧的传输时间以及用于发送维护帧等的时间之和。通信效率为 1，就意味着所有数据帧的传输时间都有效地用于传输数据帧。通信效率为 0，就意味着总线被报文的碰撞、冲突所充斥。

（2）可靠性指标

数字通信系统的可靠性可以用误码率来衡量。误码率是衡量数字通信系统可靠性的指标。它是二进制码元在数据传输系统中被传错的概率，数值近似为

$$P_{\mathrm{e}} \approx N_{\mathrm{e}} / N \tag{1-2}$$

式中，N 为传输的二进制码元总数，N_{e} 为被传输错的码元数，理论上应有 $N \to \infty$。实际使用中，N 应足够大，才能把 P_{e} 近似为误码率。

（3）通信信道的频率特性

频率特性是描述通信信道在不同频率的信号通过以后，其波形发生变化的特性。

频率特性分为幅频特性和相频特性。幅频特性指不同频率的信号通过信道后，其幅值受到不同衰减的特性；相频特性指不同频率的信号通过信道后，其相角发生不同程度改变的特性。理想信道的频率特性应该是对不同频率产生均匀的幅频特性和线性相频特性，而实际信道的频率特性并非理想。因此，通过信道后的波形会产生畸变。如果信号的频率在信道带宽范围内，则传输的信号基本上不失真，否则，信号的失真将较严重。

信道的频率特性不理想是由于传输线路并非理想线路。实际的传输线路存在电阻、电感、电容，由它们组成分布参数系统。由于电感、电容的阻抗随频率而变，故信号的各次谐波的幅值衰减不同，其相角变化也不尽相同。当然，信道的频率特性不仅与介质相关，而且和中间通信设备的电气特性有关。

（4）介质带宽

通信系统中所传输的数字信号可以分解成无穷多个频率、幅度、相角各不相同的正弦波。这就意味着传输数字信号相当于传送无数多个简单的正弦信号。信号所含频率分量的集合称为频谱。频谱所占的频率宽度称为带宽。发送端所发出的数字信号的所有频率分量都必须通过通信介质到达接收端，接收端才能再现该数字信号的精确拷贝。如果其中一部分频率分量在传输过程中被严重衰减，就会导致接收端信号变形。如果能接收到具有主要振幅的那部分分量，则仍可以按适当的精度复制出发送端所发出的数字信号。

以一定的幅度门限为依据，将在接收端能收到的那部分主要信号的频谱从原来的无穷大频谱中划分出来，便形成该信号的有效频谱。有效频谱的频带宽度称为有效带宽。

实际传输介质的带宽是有限的，它只能传输某些频率范围内的信号。一种介质只能传输有效带宽在介质带宽范围内的信号。如果介质带宽小于信号的有效带宽，信号就可能产生失真而使接收端难以正确辨认。

2．数据编码

计算机网络系统的通信任务是传送数据或数据化的信息。这些数据通常以离散的二进制0、1序列的方式表示。码元是所传输数据的基本单位。在计算机网络通信中所传输的大多为二元码，它的每一位只能在1或0两个状态中取一个，这每一位就是一个码元。

数据编码是指通信系统中以何种物理信号的形式来表达数据。用高低电平的矩形脉冲信号来表达数据的0、1状态的，称为数字数据编码。分别用模拟信号的不同幅度、不同频率、不同相角来表达数据的0、1状态的，称为模拟数据编码。

（1）数字编码波形

在设备之间传递数据就必须将数据按编码转换成适合于传输的物理信号。0、1序列码元是传输数字的基本单位，在工业数据网络通信系统中所传输的大多为二元码，它的每一位只能在1或0两状态中取一个，这每一位就是一个码元。

下面讨论几种数字数据编码波形。

1）单极性码：信号电平为单极性的编码。例如，逻辑1为高电平，逻辑0为低电平的信号表达方式，如图1-1a和图1-1c所示。

2）双极性码：信号电平为正、负两种极性的编码。例如，逻辑1为正电平，逻辑0为负电平的信号表达方式，如图1-1b和图1-1d所示。

3）归零码（RZ）：在每一位二进制信息传输之后均返回零电平的编码。例如，单极性归

零码的逻辑 1 只在该码元时间的一半维持正电平，之后也恢复到零电平，如图 1-1c 所示。图 1-1d 为双极性归零码。

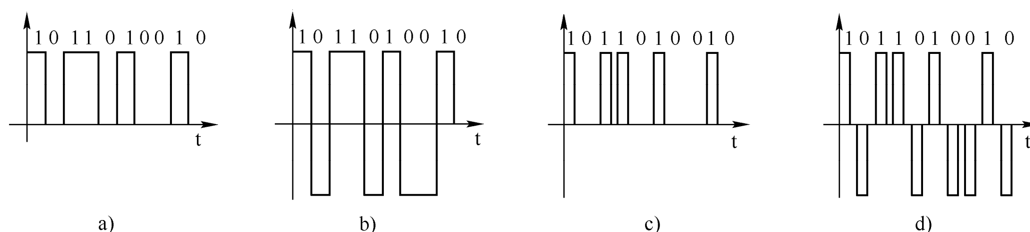


图 1-1　单、双极性的归零码和非归零码

a) 单极性非归零码　b) 双极性非归零码　c) 单极性归零码　d) 双极性归零码

4）非归零码（NRZ）。在整个码元时间都维持有效电平的编码，如图 1-1a 和图 1-1b 所示。

5）差分码。用电平的变化与否来代表逻辑“1”和“0”的编码。电平变化代表“1”，不变化代表“0”，按此规定的编码方式形成的编码称为差分码。差分码按初始状态为高电平或低电平，有相角截然相反的两种波形，其波形如图 1-2 所示。显然，差分码不可能是归零码。

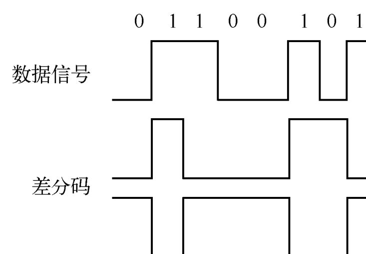


图 1-2　差分码

根据信息传输方式的不同，还可分为平衡和非平衡传输，平衡传输指无论“0”或“1”都是传输格式的一部分；而在非平衡传输中，只有“1”被传输，“0”则以指定时刻没有脉冲来表示。

6）曼彻斯特编码（Manchester encoding）是在工业数据通信中最常用的一种基带信号编码。它具有内在的时钟信息，从而能使网络上的每个节点保持时钟同步。在曼彻斯特编码中，时间被划分为等间隔的小段，其中每小段代表一个比特（即一位）。每个比特时间又被分为两半：前半个时间段所传信号是该时间段传送比特值的反码；后半个时间段传送的是比特值本身。因此从主电平跳变到低电平表示 0；从低电平跳到高电平表示 1。可见，在一个时间段内，其中间点总有一次信号电平的变化，因而携带有信号传送的同步信息。这样就无需另外传送同步信号。

差分曼彻斯特编码（differential Manchester encoding）是曼彻斯特编码的一种变形。它既具有曼彻斯特编码在每个比特时间间隔中间信号一定会发生跳变的特点，也具有差分码用电平变化代表逻辑“1”，不变化代表逻辑“0”的特点。它通过检查信号在每个周期起点处有无跳变来区分 0 和 1，边界开始有跳变代表逻辑“0”，边界开始没有跳变代表逻辑“1”，它的每一位中间的跳变仅用来提供时钟信息。这种检查信号跳变的方式往往更可靠。即使作为通信传输介质的两条导线颠倒了，对该编码信号的状态判别结果依然有效。差分曼彻斯特编码的信号波形如图 1-3 所示。

（2）模拟数据编码

模拟数据编码采用模拟信号来表示数据的 0、1 状态。信号的幅度、频率、相角描述模拟信号的参数，通过改变这 3 个参数实现模拟数据编码。幅值键控（Amplitude-Shift Keying，

ASK）、频移键控（Frequency-Shift Keying，FSK）和相移键控（Phase-Shift Keying，PSK）是模拟数据编码的 3 种编码方法。

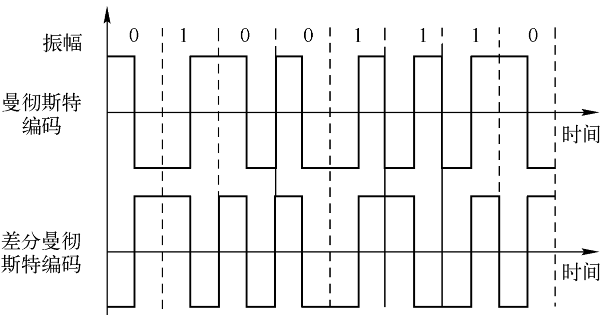


图 1-3　曼彻斯特编码与差分曼彻斯特编码的信号波形

1）在 ASK 中，载波信号的频率、相角不变，幅度随调制信号变化。图 1-4a 表示幅值键控调制后的波形与数据信号的关系。例如一个二进制数字信号在调制后波形的时域表达式：

$$S_A = a_n A\cos\omega_c t \qquad (1\text{-}3)$$

式中，A 为载波信号幅度；ω_c 为载波频率；a_n 为二进制数字 0 或 1。

2）在 FSK 中，载波信号的频率随着调制信号而变化，而载波信号的幅度、相角不变。例如在二进制频移键控中，可定义信号 0 对应的载波频率大，信号 1 对应的载波频率小，调制后信号波形如图 1-4b 所示。现场总线的 HART 通信信号即采用了这种编码方式，其信号频率为 1 200Hz 时表示 1，信号频率为 2 200Hz 时表示 0。

3）在 PSK 中，载波信号的相角随着调制信号而变化，而载波信号的幅度、频率不变。例如在二进制相移键控中，通常用相应的 0° 和 180° 来分别表示 1 或 0，调制后信号的典型波形如图 1-4c 所示。

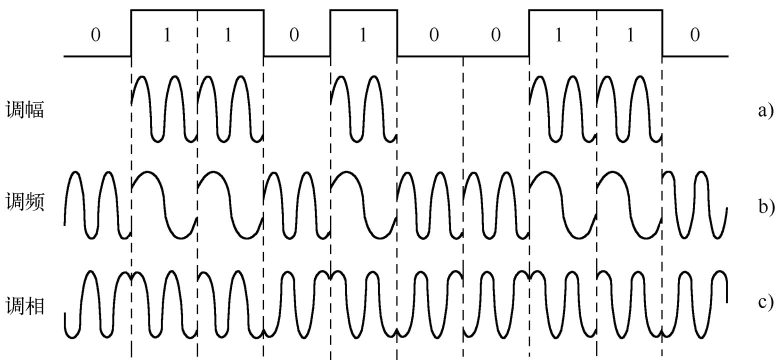


图 1-4　模拟数据编码调制的信号波形

信号调制的实现方法：调幅（AM）——幅值键控法，调频（FM）——频移键控法，调相（PM）——相移键控法。

频移键控法调制实现的示例如图 1-5 所示。发送端将数字信号调制成不同频率的模拟信

号；接收端通过解调，将模拟信号还原成数字信号。

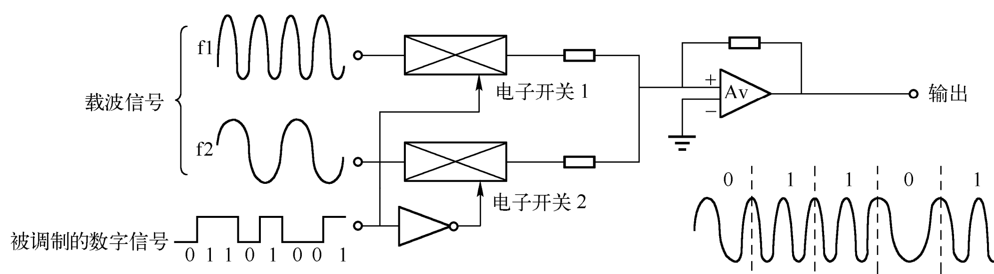


图 1-5　频移键控法调制的实现原理示意图

3．数据传输方式

数据传输方式是指数据代码的传输顺序和数据信号传输时的同步方式，有串行传输与并行传输，同步传输与异步传输，位同步、字符同步与帧同步等几种。

（1）串行传输和并行传输

在串行传输中，数据流以串行方式逐位地在一条信道上传输。每次只能发送一个数据位，发送方必须确定是先发送数据字节的高位还是低位。同样，接收方也必须知道所收到的字节的第一个数据位应处于什么位置。串行传输具有易于实现，在长距离连接中可靠性高等优点。适合远距离的数据通信，但需要在收发双方采取同步措施。

并行传输是将数据以成组的方式在两条以上的并行通道上同时传输。它可以同时传输一组数据位，每个数据位使用单独的一条导线，例如采用 8 条导线并行传输一个字节的 8 个数据位。另外用一条选通线通知接收者接收该字节，接收方可对并行通道上各条导线的数据信号并行取样。若采用并行传输进行字符通信，则不需要采取特别措施就可以实现收发双方的字符同步。

串行传输和并行传输的区别在于组成一个字符或字节的各数据位是依顺序逐位传输还是同时并行地传输。

（2）同步传输与异步传输

在数据通信系统中，各种处理工作总是在一定的时序脉冲控制下进行的，而收发端工作的协调一致性又是实现信息传输的关键，这也就是数据通信的传输同步问题。

串行数据传输中的二进制代码在一条总线上以数据位为单位按时间顺序逐位传送，接收端按顺序逐位接收，接收端必须能正确地按位区分才能正确恢复所传输的数据。串行通信中的发送者和接收者都需要使用时钟信号，通过时钟决定什么时候发送和读取每一位数据。

同步就是接收端要按发送端所发送的每个码元的重复频率以及起止时间来接收数据。在通信时，接收端要校准自己的时间和重复频率，以便和发送端取得一致，这一过程称为同步过程。同步是数字通信系统中必须解决的一个重要问题。常用信息传输的同步方式有同步传输方式和异步传输方式。

1）同步传输方式：同步传输方式各字符没有起始位和停止位，采用位同步技术。位同步就是使接收端接收的每一位数据信息都要和发送端准确地保持同步，实现位同步方法有外同步和自同步。

外同步法是在发送数据前，发送端先向接收端发一串同步的时钟。

自同步法是从数据信号波形本身提取同步信号的方法，时钟信号和传输信息同时传输到接收端。如数字信号采用曼彻斯特编码和差分曼彻斯特编码，这两种编码本身都是自同步编码。

由于同步方式比异步式传输效率高，适用于高速传输要求，一般在高速传输数据的系统中采用同步方式。

2）异步传输方式：异步方式又称起止式（start-stop）同步方式，这是在计算机通信中常用的同步方式。异步方式中，并不要求收发两端在传送代码的每一比特（位）时都同步。例如在字符同步的异步方式传输中，在传输的字符前，设置一个启动用的起始位，预告字符的信息代码即将开始，在信息代码和校验信号（一般总共为 8bit）结束后，也设置 1～2bit 的终止位，表示该字符已结束。终止位也反映了平时不进行通信的状态。当从不传输信息状态转到起始位状态时，在接收端将检测出极性状态的改变，利用这种改变，就可启动定时机构，实现同步。接收端收到终止位，就将定时机构复位，准备接收下一个字符代码。

4．通信线路的工作方式

（1）单工通信

单工是指所传送的信息始终朝着一个方向，而不进行与此相反方向的传送，如图 1-6a 所示。设 A 为发送终端，B 为接收终端，数据只能从 A 传送至 B，而不能由 B 传送至 A。单工通信线路一般采用二线制。

（2）半双工通信

半双工通信是指信息流可在两个方向上传输，但同一时刻只限于一个方向传输，如图 1-6b 所示。信息可以从 A 传至 B，或从 B 传至 A，所以通信双方都具有发送器和接收器。要实现双向通信必须改换信道方向。半双工通信采用二线制线路，当 A 站向 B 站发送信息时，A 站将发送器连接在信道上，B 站将接收器连接在信道上；而当 B 站向 A 站发送信息时，B 站则要将接收器从信道上断开，并把发送器接入信道，A 站也要相应地将发送器从信道上断开，而把接收器接入信道。这种在一条信道上进行转换，实现 A→B 与 B→A 两个方向通信的方式，称为半双工通信。

（3）全双工通信

全双工通信是指通信系统能同时进行如图 1-6c 所示的双向通信。它相当于把两个相反方向的单工通信方式组合在一起。这种方式常用于计算机与计算机之间的通信。

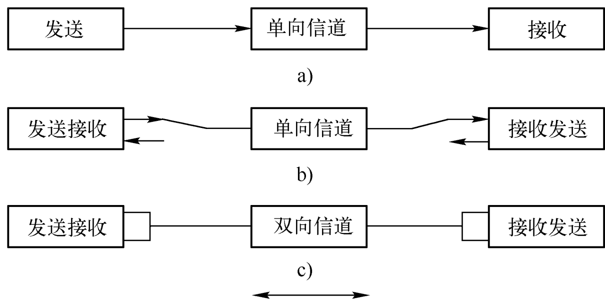


图 1-6　三种通信线路的工作方式

a) 单工通信　b) 半双工通信　c) 全双工通信

5．信号的传输模式

（1）基带传输

基带传输就是在数字通信的信道上直接传送数据的基带信号，即按数据波的原样进行传输，不包含有任何调制，它是最基本的数据传输方式。

目前大部分微机局域网，包括控制局域网，都是采用基带传输方式的基带网。基带网的特点如下：信号按位流形式传输，整个系统不用调制解调器，这使得系统价格低廉。它可采用双绞线或同轴电缆作为传输介质，也可采用光缆作为传输介质。与宽带网相比，基带网的传输介质比较便宜，可以达到较高的数据传输速率（一般为 1M～10Mbit/s），但其传输距离一般不超过 25km，传输距离越长，质量越低。基带网中线路工作方式只能为半双工方式或单工方式。

（2）载波传输

载波传输：载波传输采用数字信号对载波进行调制后实行传输。最基本的调制方式有幅值键控（ASK）、频移键控（FSK）、相移键控（PSK）三种。

在载波传输中，发送设备首先要产生某个频率的信号作为基波来承载信息信号，这个基波就称为载波信号，基波频率就称为载波频率；然后按幅值键控、频移键控、相移键控等不同方式改变载波信号幅值、频率、相角，形成调制信号后发送。

（3）宽带传输

由于基带网不适于传输语言、图像等信息。宽带传输与基带传输的主要区别为：一是数据传输速率不同，基带网的数据传输速率范围为 0～10Mbit/s，宽带网可达 0～400Mbit/s；二是宽带网可划分为多条基带信道，提供良好的通信路径。一般将宽带局域网与有线电视系统共建，以节省投资。

（4）异步传输模式

异步传输模式（Asynchronous Transfer Mode，ATM）是一种新的传输与交换数字信息技术，也是实现高速网络的主要技术。它支持多媒体通信，包括数据、语音和视频信号，按需分配频带，具有低延迟特性，速率可达 155kbit/s 到 2.4Mbit/s，也有 25Mbit/s 和 50Mbit/s 的 ATM 技术，可适用于局域网和广域网。

6．差错控制

为了提高通信系统的传输质量而提出的有效地检测错误，并进行纠正的方法叫做差错检测和校正，简称为差错控制。差错检测是让报文分组中包含使接收端发现差错的冗余信息，但它不能确定是哪一位出错，自己也不能纠正传输中的差错；而差错纠正是让每个传输的报文分组中带有足够的冗余信息，以便接收端能发现并自动纠正传输错误。差错检测原理简单，实现容易，编码与解码速度快，目前正得到广泛应用。

（1）差错的检测方法

差错检测就是监视接收到的数据并判别是否发生了传输错误。差错检测并不识别哪个或哪些位出现了错误，仅仅识别出错误的出现。差错检测最常用的方法如下：

1）冗余：冗余是对每个字符都传输两次。如果连续两次没有收到相同的字符，就意味着发生了一个传输错误。

2）回送：回送被用在操作人员手工从键盘输入数据的通信系统中。把接收端收到的每一个字符都回送给操作人员，让操作人员来确认字符是否被正确输入了。如果在回送字符期间

出现了传输错误，就要进行重复传输。

3）精确计数编码：利用精确计算数编码时，在每个字符中的“1”的数量是相同的。接收端计算一个字符中“1”的个数，如果其总数不等于预先设定的值，就表明发生了一个错误。

4）奇偶校验：在奇偶校验中，一个单一的位（奇偶校验位）被加在每个字符上，以使得一个字符中“1”的总数要么是奇数（奇校验），要么是偶数（偶校验）。奇偶校验有可能漏掉大量的错误，但应用起来非常简单。

5）求校验和：在发送端和接收端都对传输的数据进行求和操作，在发送端将校验和附加在数据信息之后。如果接收端的校验和与发送端的校验和不同，就表明发生了错误。和校验方法能检测出 95%的错误，但与奇偶校验方法相比，增加了计算量。

6）循环冗余校验（Cyclic Redundancy Check，CRC）：循环冗余校验对传输序列进行一次除法操作，将进行除法操作的余数附加在传输信息的后边。在接收端，也进行同样的除法过程。如果接收端除法过程的结果不是零，就表明发生了一个错误。CRC 错误检查方法能够检测出大约 99.95%的错误，但计算量大。

（2）差错的纠正方法

差错纠正的两种最常用的方法如下：

1）重新传输：当检测到一个错误时，接收端自动地请求重新传输这条信息，这种方法经常被称做 ARQ，即自动重传，ARQ 技术很简单，但可能会因确认和重发造成通信障碍。

2）前向差错纠正（FEC）：FEC 技术在接收端检测和纠正差错，不需要请求重新发送。利用 FEC 技术将一些位加入到通信序列中，这些额外的位按照某种方式进行编码，这种方式允许对每条信息检测到一定数量的错误并进行纠正。

1.2.2 网络拓扑

网络的拓扑结构是指网络中节点的互连形式。在网络拓扑结构中，星形、环形、总线型和树形较为常见。

1．环形拓扑

图 1-7 为环形拓扑的连接示意图。在环形拓扑中，网络中有许多中继器进行点对点链路连接，构成一个封闭的环路。中继器接收前驱站发来的数据，然后按原来速度一位一位地从另一条链路发送出去。链路是单向的，数据沿一个方向（顺时针或逆时针）在网上环行。每个工作站通过中继器再连至网络。一个站发送数据，按分组进行，数据拆成分组加上控制信息插入环上，通过其他中继器到达目的站。

2．星形拓扑

图 1-8 为星形拓扑的连接示意图。在星形拓扑中，每个站通过点对点连接到中央节点，任何两站之间通信都通过中央节点进行。一个站要传送数据，首先向中央节点发出请求，要求与目的站建立连接。连接建立后，该站才向目的站发送数据。这种拓扑采用集中式通信控制策略，所有通信均由中央节点控制，中央节点必须建立和维持许多并行数据通路，因此中央节点的结构显得非常复杂，而每个站的通信处理负担很小，只需满足点对点链路简单通信要求，结构很简单。

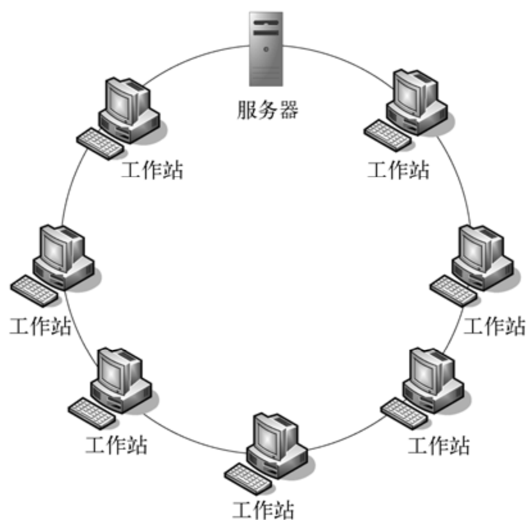


图 1-7　环形拓扑的连接示意图

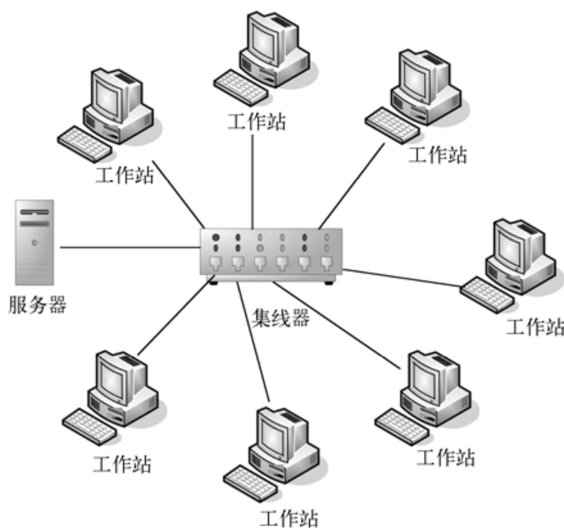


图 1-8　星形拓扑的连接示意图

3．总线型拓扑

图 1-9 为总线型拓扑的连接示意图。在总线型拓扑中，传输介质是一条总线，工作站通过相应硬件接口接至总线上。一个站发送数据，所有其他站都能接收。

4．树形拓扑

树形拓扑是总线型拓扑的扩展形式，传输介质是不封闭的分支电缆。它和总线型拓扑一样，一个站发送数据，其他站都能接收。因此，总线型和树形拓扑的传输方式称作多点式或广播式。图 1-10 为树形拓扑的连接示意图。

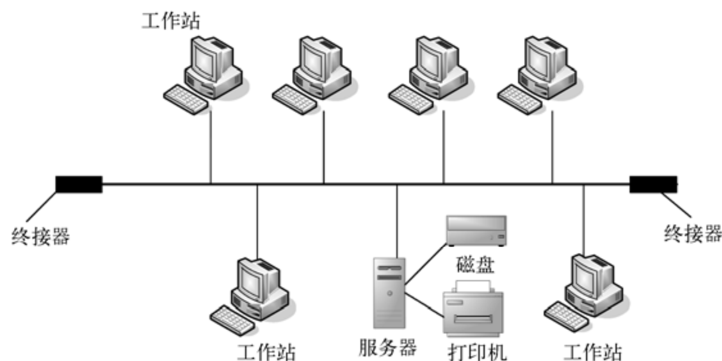


图 1-9　总线型拓扑的连接示意图

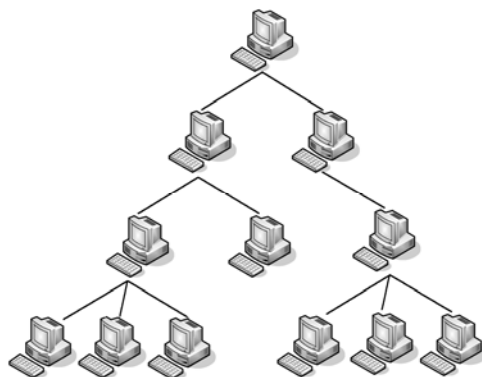


图 1-10　树形拓扑的连接示意图

1.2.3　网络的传输介质

传输介质是网络中连接收发双方的物理通路，也是通信中实际传送信息的载体。网络中常用的传输介质有电话线、同轴电缆、双绞线、光导纤维电缆、无线与卫星通信。传输介质的特征对网络中数据通信质量影响很大。

1．双绞线

无论对于模拟数据还是对于数字数据信号，双绞线都是最常用的传输介质，其主要特征如下：

（1）物理特性

双绞线由按规则螺旋结构排列的 2 根或 4 根绝缘导线组成。一对线可以作为一条通信线路，各线对螺旋排列的目的是使各线对之间的电磁干扰最小。

（2）传输特性

双绞线最普遍的应用是语音信号的模拟传输。一条全双工音频通道的标准带宽是 300～3400Hz。

（3）连通性

双绞线可用于点对点连接，也可用于多点连接。

（4）地理范围

双绞线用作远程中继线时，最大距离可达 15km；用于 10Mbit/s 局域网时，与集线器的最大距离为 100m。

（5）抗干扰性

双绞线的抗干扰性取决于相邻线对的扭曲长度及屏蔽。在低频传输时，其抗干扰能力相当于同轴电缆；在频率为 10～100kHz 时，其抗干扰能力低于同轴电缆。

2．同轴电缆

同轴电缆是网络中应用十分广泛的传输介质之一。其主要特性如下：

（1）物理特性

同轴电缆结构如图 1-11 所示。它由内导体、外导体、绝缘层及外部保护层组成。同轴介质的特性参数由内、外导体及绝缘层的电气参数和机械尺寸决定。

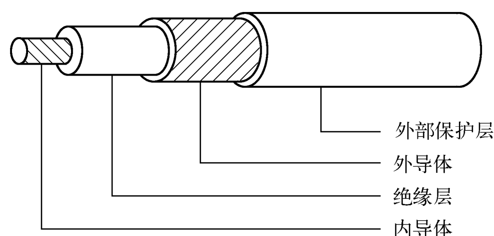


图 1-11　同轴电缆结构示意图

（2）传输特性

根据同轴电缆的带宽，同轴电缆可以分为基带同轴电缆和宽带同轴电缆两类。基带同轴电缆一般仅用于数字数据信号传输；宽带同轴电缆可以使用频分多路复用方法，将一条宽带同轴电缆的频带划分成多条通信信道，以支持多路传输。

（3）连通性

同轴电缆支持点对点连接，也支持多点连接。宽带同轴电缆可支持数千台设备的连接；基带同轴电缆可支持数百台设备的连接。

（4）地理范围

基带同轴电缆的最大距离在几千米范围内；而宽带同轴电缆的最大距离可达几十千米。

（5）抗干扰性

同轴电缆的结构使得它的抗干扰能力较强。

（6）价格

同轴电缆的造价介于双绞线与光缆之间，维护方便。

3．光缆

光缆是光导纤维构成的线缆，它是网络传输介质中性能最好、应用最广泛的一种。

（1）物理特性

光纤是直径为 50～100μm 的能传导光波的柔软介质。有玻璃和塑料材质的光纤，用超高纯度石英玻璃纤维制作的光纤的传输损耗很低。把折射率较高的单根光纤用折射率较低的材质包裹起来，就可以构成一条光纤通道，多条光纤组成一束就构成光缆。光缆的结构如图 1-12a 所示。

（2）传输特性

光导纤维通过内部的全反射来传输一束经过编码的光信号。光波通过光导纤维内部的全反射进行光传输的过程如图 1-12b 所示。

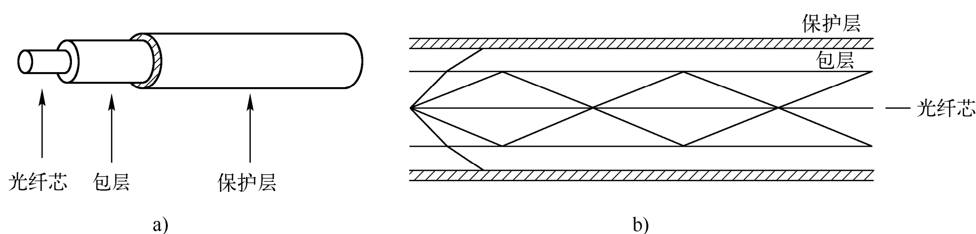


图 1-12　光缆结构示意

a) 光缆结构　b) 光纤内部的光传输过程

由于光纤的折射系数高于外层的折射系数，所以可以形成光波在光纤与包层界面上的全反射，以小角度进入的光波沿纤维按全反射方式向前传播。

典型光纤传输系统的结构如图 1-13 所示。光纤发送端采用两种光源：发光二极管 LED 和注入型激光二极管。在接收端将光信号转换成电信号时使用光敏二极管检波器。光纤传输速率每秒可达几千兆位。

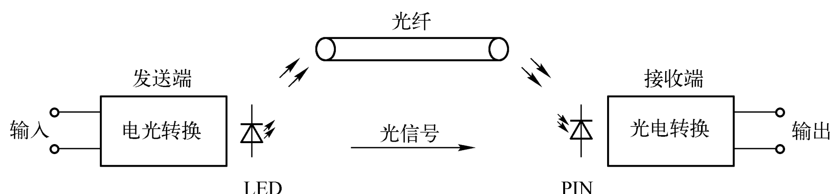


图 1-13　光缆系统传输示意图

（3）连通性

光纤最普通的连接方法是点对点方式，在某些系统中也采用多点连接方式。

（4）地理范围

光纤信号衰减极小，它可以在 6～8km 距离内不使用中继器实现高速率数据传输。

（5）抗干扰性

光纤不受外界电磁干扰与噪声的影响，能在长距离、高速率中保持低误码率。双绞线典型的误码率在 10^{-5}～10^{-6} 之间，基带同轴电缆为 10^{-7}，宽带同轴电缆为 10^{-9}，而光纤误码率可以低于 10^{-10}。此外，光纤传输的安全性与保密性也很好。

（6）价格

光纤价格高于同轴电缆与双绞线。由于光纤具有低损耗、宽频带、高数据传输速率、低误码率、安全保密性好等特性，因此，它是一种最有前途的传输介质。

4．无线通信

无线通信主要有微波通信、红外通信与激光通信。卫星通信可以看成是一种特殊的微波通信系统。微波载波频率是 2～40GHz。由于微波载波频率很高，可以同时传送大量信息，例如，一个带宽为 2MHz 的微波频段可以容纳 500 路语言信道。当用于数字通信时，其数据传

输速率也很高。微波属于一种视距传输，它沿直线传播，不能绕射。红外通信与激光通信也属于方向性极强的直线传播，发送端与接收端必须可以直视，中间没有阻挡。由于微波通信信道、红外通信信道与激光通信信道都不需要铺设电缆，所以，对于连接不同建筑物之间的局域网特别有用。

1.2.4　网络传输的介质访问控制方式

在总线型和环形拓扑中，网上设备必须共享传输线路。为避免在同一时间有几个设备同时争用传输介质，需有某种介质访问控制方式，以便协调各设备访问介质的顺序。

通信中对介质的访问可以是随机的，即各工作站可在任何时刻，任意地访问介质，也可以是受控制的，即各工作站可用一定的算法调整各站访问介质的顺序和时间。在随机访问方式中，常用的争用总线技术为 CSMA/CD。在控制访问方式中，常用的争用总线技术为令牌总线、令牌环，或称之为标记总线、标记环。另外争用总线还有异步分时复用控制方式。

1．载波监听、多路访问和冲突检测控制方式

这种控制方式对任何工作站都没有预约发送时间。工作站的发送是随机的，必须在网络上争用传输介质。若同一时刻有多个工作站向传输线路发送信息，则这些信息会在传输线上相互混淆而遭破坏，称为“冲突”。为尽量避免由于竞争引起的冲突，每个工作站在发送信息之前，都要监听传输线上是否有信息在发送，这就是“载波监听”。

载波监听 CSMA 的控制方案是先听再讲。一个站要发送，首先需监听总线，以决定介质上是否存在其他站的发送信号。如果介质是空闲的，则可发送。如果介质是忙的，则等待一段随机时间，重复第一步。

由于传输线上不可避免的有传输延迟，有可能多个站同时监听到线上空闲并开始发送，从而导致冲突。故每个工作站发送信息之后，要继续监听线路，判定是否有其他站正与本站同时向传输线发送，一旦发现，便中止当前发送，这就是“冲突检测”。

载波监听多路访问/冲突检测的协议，简写为 CSMA/CD，已广泛用于局域网中。每个站在发送帧期间，同时有检测冲突的能力，即所谓边讲边听。一旦检测到冲突，就立即停止发送，并向总线上发一串阻塞信号，通知总线上各站冲突已发生，这样通道容量不致因传送已损坏的帧而白白浪费。

2．令牌控制方式

CSMA 的访问存在报文冲突问题，产生冲突的原因是由于各站点的报文发送是随机的。为了解决冲突问题，可采用有控制的发报方式，令牌方式是一种按一定顺序在各站点传递令牌（token）的方法。谁得到令牌，谁才有发报权。令牌访问原理可用于环形网络，构成令牌环形网；也可用于总线网，构成令牌总线网络。

（1）令牌环方式

令牌是环形结构局域网采用的一种访问控制方式。由于在环形结构网络上，某一瞬间可以允许发送报文的站点只有一个，令牌在网络环路上不断地传送，只有拥有此令牌的站点，才有权向环路上发送报文，而其他站点仅允许接收报文。站点在发送完毕后，便将令牌交给网络上的下一个站点，如果该站点没有报文需要发送，便把令牌顺次传给下一个站点。所以，表示发送权的令牌在环形信道上不断循环。环上每个相应站点都可获得发报权，而任何时刻只会有一个站点利用环路传送报文，因而在环路上保证不会发生访问冲突。

（2）令牌总线方式

令牌总线方式和 CSMA/CD 方式一样，采用总线型网络拓扑，但不同的是令牌总线方式在网络上各工作站按一定顺序形成一个逻辑标记环。每个工作站在环中均有一个指定的逻辑位置，末站的后站就是首站，即首尾相连。每站都了解先行站和后继站地址，总线上各站的物理位置与逻辑位置无关。

像令牌环方式那样，令牌传递总线方式也具备称为令牌的控制帧和调整访问的权力。收到令牌的站点在一段规定时间内被授予对介质的控制权，因而该站可以发送一帧或多帧信息。当站传输已经完成或时间已到时，它就将令牌传递到逻辑环中的下一工作站。网上的工作站点也可以退出环成为非活动站点，这些站点仅能响应询问或请求应答。由于只有收到令牌的站点才能将信息帧送到总线上，所以，它不同于 CSMA/CD 访问方式，不可能产生冲突。因此，标记环的信息帧长度只需根据要传送的信息长度来确定。

3．分时复用

对每个节点预先分配好特定的一段时间，让每个节点在这段时间内占有总线，这种多个节点按划分的时间顺序占有总线的工作方式称为分时多路复用。比如让节点 A、B、C、D 分别按 1、2、3、4 的顺序占用总线。如果事先可以预计每个节点占用总线的时间、需要的通信时间或传送的报文字节数量，则可以准确估算出每个节点两次占用总线之间的循环周期。这在控制网络的应用条件下满足某种确定的时间要求是很有用的。

1.2.5　现场总线通信模型

一个通用的网络通信参考模型，需要解决各方面可能遇到的问题，需要具备丰富的功能。作为工业控制现场底层网络的现场总线，要构成开放互连系统，应该如何选择通信模型，是采用完全型还是简化型，是否需要实现 OSI 的全部功能，是否要采用那样复杂的协议，具有七层 OSI 参考模型是否适应工业现场的通信环境，这是现场总线技术形成的过程中必须考虑的重要问题。

工业生产现场存在大量传感器、控制器、执行器等，它们通常相当零散地分布在一个较大范围内。对由它们组成的工业控制底层网络来说，单个节点面向控制的信息量不大，信息传输的任务相对比较简单，但实时性、快速性的要求较高。如果按照七层模式的参考模型，由于层间操作与转换的复杂性，网络接口的造价与时间开销显得过高。为满足实时性要求，也为了实现工业网络的低成本，现场总线采用的通信模型大都在 OSI 模型的基础上进行了不同程度的简化。

典型的现场总线协议模型采用 OSI 模型中的三个典型层：物理层、数据链路层和应用层，在省去中间 3～6 层后，考虑现场总线的通信特点，设置一个现场总线访问子层。它具有结构简单、执行协议直观、价格低廉等优点，也满足工业现场应用的性能要求。它是 OSI 模型的简化形式，其流量与差错控制在数据链路层中进行。因而与 OSI 模型不完全保持一致。总之，开放系统互连模型是现场总线技术的基础。现场总线参考模型既要遵循开放系统集成的原则，又要充分兼顾测控应用的特点和特殊要求。

如 Profibus 参考模型采用了 OSI 模型的物理层、数据链路层，隐去了第 3～7 层。而 CAN 也采用了 ISO/OSI 模型全部七层中的两层，物理层和数据链路层。其中物理层完成电气连接、实现驱动器/接收器特性、定时、同步、位编码解码。数据链路层分为逻辑链路控制与媒体访

问控制两部分，分别完成接收滤波、超载通知、恢复管理，以及应答、帧编码、数据封装拆装、媒体访问管理、出错检测等。

1.3 CAN 总线基础知识

1.3.1 CAN 总线的发展历程

从 1980 年开始，德国 Bosch 公司的工程师就开始论证串行总线用于客车系统的可行性。因为没有一种现成的网络方案能够完全满足汽车工程师们的要求，于是，在 1983 年初，Uwe Kiencke 开始研究一种新的串行总线。新总线的主要方向是增加新功能、减少电气连接线。来自 Mercedes-Benz 的工程师较早制定了总线的状态说明，而 Intel 也准备作为半导体生产的主要厂商。当时聘请的顾问之一是来自于德国 Braunschweig-Wolfenbüttel 的 Applied Science 大学的教授 Wolfhard Lawrenz 博士，他给出了新网络方案的名字“Controller Area Network”，简称 CAN。来自 Karlsruhe 大学的教授 Horst Wettstein 博士也提供了理论支持。

1986 年 2 月 Robert Bosch 公司在 SAE 汽车工程协会大会上介绍了一种新型的串行总线 CAN 控制器局域网，标志着 CAN 总线的诞生，在该大会上，由 Bosch 公司研究的新总线系统被称为“汽车串行控制器局域网”。Uwe Kiencke、Siegfried Dais 和 Martin Litschel 分别介绍了这种多主网络方案。此方案基于非破坏性的仲裁机制，能够确保高优先级报文的无延迟传输。并且，不需要在总线上设置主控制器。此外，上述几位教授和 Bosch 公司的 Wolfgang Borst、Wolfgang Botzenhard、Otto Karl、Helmut Schelling、Jan Unruh 已经实现了数种在 CAN 中的错误检测机制。该错误检测也包括自动断开故障节点功能，以确保能继续进行剩余节点之间的通信。传输的报文并非根据报文发送器/接收器的节点地址识别（几乎其他的总线都是如此），而是根据报文的内容识别。同时，用于识别报文的标识符也规定了该报文在系统中的优先级。

当关于这种革新的通信方案的大部分文字内容制定之后，于 1987 年中期，Intel 比计划提前两个月交付了首枚 CAN 控制器：82526，这是 CAN 方案首次通过硬件实现。仅仅用了四年的时间，设想就变成了现实。不久之后，Philips 半导体推出了 82C200。

1990 年 Bosch CAN 规范（CAN 2.0 版）被提交给国际标准化组织。在数次行政讨论之后，应一些主要的法国汽车厂商要求，增加了“Vehicle Area Network（VAN）”内容，并于 1993 年 11 月出版了 CAN 的国际标准 ISO 11898。除了 CAN 协议外，它也规定了最高至 1Mbit/s 波特率时的物理层。同时，在国际标准 ISO 11519-2 中也规定了 CAN 数据传输中的容错方法。1995 年，国际标准 ISO 11898 进行了扩展，以附录的形式说明了 29bit CAN 标识符。

今天在欧洲几乎每一辆新客车均装配有 CAN 局域网。同样 CAN 也用于其他类型的交通工具，从火车到轮船或者用于工业控制。CAN 已经成为全球范围内最重要的总线之一，甚至领导着串行总线。在 1999 年接近 6 千万个 CAN 控制器投入应用，2000 年市场销售超过 1 亿个 CAN 器件。

CAN 总线由于其高性能、高可靠性、实时性等优点现已广泛应用于工业自动化、多种控制设备、交通工具、医疗仪器以及建筑、环境控制等众多部门，并且在我国迅速普及推广。

1.3.2　CAN 总线的通信方式

1．“载波监测，多主掌控/冲突检测”（CSMA/CD）的通信技术

CAN 采用一种叫做“载波监测，多主掌控/冲突检测”（CSMA/CD）的通信方式。允许在总线上的任一设备有同等的机会取得总线控制权来向外发送信息。如果在同一时刻有两个以上的设备欲发送信息，就会发生数据冲突，CAN 总线能够实时地检测这些冲突情况，并作出相应的仲裁而不会破坏待传的信息。

“载波监测”是指在总线上的每个节点在发送信息报文前都必须监测到总线上有一段时间的空闲状态。一旦空闲状态被监测到，那么每个节点都有均等的机会来发送报文，这被称作“多主掌控”。

“多主掌控/冲突检测”是每个节点都可以主动发送数据，但是如果两个以上节点同时发送信息时，节点本身首先会检测到出现冲突，然后采取相应的措施来解决这一冲突情况。此时优先级高的报文先发送，优先级低的报文发送会暂停。在 CAN 总线协议中是通过一种非破坏性的位仲裁方式来实现冲突检测的，这也就意味着当总线出现发送冲突时，通过仲裁后原发送信息不会受到任何影响。所有的仲裁都不会破坏优先级高的报文信息内容，也不会对其发送产生任何的延时。

2．基于报文的通信技术

CAN 总线采用的是一种基于报文而不是基于节点地址的通信方式，也就是说报文不是按照地址从一个节点传送到另一个节点。这就允许不同的信息以“广播”的形式发送到所有节点，并且可以在不改变信息格式的前提下对报文进行不同配置。CAN 总线上报文所包含的内容只有优先级标志区和欲传送的数据内容。所有节点都会接收到在总线上传送的报文，并在正确接收后发出应答确认。至于该报文是否要做进一步的处理或被丢弃将完全取决于接收节点本身。同一个报文可以发送给特定的节点或许多节点，设计者可以根据要求来设计相应的网络系统。

CAN 总线协议另外一个有用的特性是一个站点可以主动要求其他站点发送信息，这种特性叫做“远程终端发送请求（RTR）”。站点并不等待信息的到来，而是主动去索取。

例如，汽车的中央安全系统会频繁地更新一些像安全气袋等关键传感器的信息，但是有些信息如油压传感器或电池电压传感器的信息系统可能不会经常收到。为了确保了解这些设备是否工作正常，系统必须定期地要求此类设备发送相关的信息以便检查整个系统的工作情况。设计人员就可以利用这种“远程终端发送请求”特性来减少网络的数据通信量，同时维持整个系统的完整性。基于报文的这种协议另外一个好处是新的站点可以随时方便地加入到现有的系统中，而不需对所有站点进行重新编程以便它们能识别这一新站点。一旦新站点加入到网络中，它就开始接收信息，判别信息标识，然后决定是否作处理或直接丢弃。

CAN 总线定义了四种报文用于总线通信。第一种称为“数据帧”；第二种称为“远程帧”，用于一个站点主动要求其他站点发送信息；另外两种用于差错处理，分别叫做“出错帧”和“超载帧”。如果站点在接收过程中检测到任意在 CAN 总线协议中定义了的错误信息，它就会发送一个出错帧；当一个站点正忙于处理接收的信息，需要额外的等待时间接收下一报文时，可以发送超载帧，通知其他站点暂缓发送新报文。

3．高速且具备复杂的错误检测和恢复能力的高可靠通信技术

CAN 总线协议有一套完整的差错定义，能够自动地检测出这些错误信息，由此保证了被传信息的正确性和完整性。CAN 总线上的每个节点具有检测多种通信差错信息的能力并采取相关的应对措施：发送错误可通过“CRC 出错”检测到；普通接收错误可通过“应答出错”检测到；CAN 报文格式错误可通过“格式出错”检测到；CAN 总线信号错误可通过“位出错”检测到；同步和定时错误可通过“阻塞出错”检测到。每个 CAN 总线上的节点都有一个出错计数器用以记录各种错误发生的次数。通过这些计数器可以判别出错的严重性，确认这些节点是否应工作在降级模式；总线上的节点可以从正常工作模式（正常收发数据和出错信息）降级到消极工作模式（只有在总线空闲时才能取得控制权），或者到关断模式（和总线隔离）。CAN 总线上各节点还有能力监测是短期的干扰还是永久性的故障，并采取相关的应对措施，这种特性被叫做“故障界定隔离”。采取了这种故障界定隔离措施后，故障节点将会被及时关断，不会永久占用总线。这一点对关键信息能在总线上畅通无阻地传送是非常重要的。

1.3.3　CAN 总线的技术特点

CAN 属于总线式串行通信网络，由于其采用了许多新技术及独特的设计，与一般的通信总线相比，CAN 总线的数据通信具有可靠性高、实时性和灵活性强等优点，具体概括如下：

1）CAN 为多主工作方式，网络上任意节点均可在任意时刻主动向网络上其他节点发送信息，而不分主从，通信方式灵活，且无需节点地址等节点信息。利用这一特点可方便构成多机备份系统。

2）CAN 的节点信息分为不同的优先级，可满足不同的实时要求，高优先级的数据最多可在 134μs 内得到传输。

3）CAN 采用非破坏性的总线仲裁技术，当多个节点同时向总线发送信息时，优先级较低的节点会主动退出发送，而高优先级的节点可不受影响地继续传输数据，从而大大节省了总线冲突仲裁的时间。

4）CAN 通过报文滤波即可实现点对点、一点对多点及全局广播等几种方式传送和接收数据，无需专门的“调度”。

5）CAN 的一个节点可以主动要求其他节点发送信息，这种特性叫做“远程终端发送请求（RTR）”。

6）CAN 的直线通信距离最长可达 10km（速率 5kbit/s 以下），通信速率最高可达 1Mbit/s（此时通信距离最长为 40m）。

7）CAN 上的节点数主要取决于总线驱动电路，目前可达 110 个；报文标识符可达 2032 种（CAN2.0A），而扩展标准（CAN2.0B）的报文标识符几乎不受限制。

8）采用短帧结构，传输时间短，受干扰概率低，具有良好的检错效果。

9）CAN 的每帧信息都有 CRC 校验及其他检错措施，保证了数据出错率极低。

10）CAN 的通信介质可为双绞线、同轴电缆或光纤，选择灵活。

11）CAN 节点在错误严重的情况下具有自动关闭输出的功能，以使总线上其他节点的操作不受影响。

1.4　本章小结

现场总线可以实现测控设备之间的通信，是控制系统网络化和信息化的基础。目前现场总线技术已广泛应用于工业、农业以及军事领域。本章介绍了现场总线的概念、产生、技术特点、标准形成和技术基础，以及 CAN 总线的基础知识。

思考题与习题

1.1　简要说明现场总线定义。
1.2　简要分析现场总线技术特点。
1.3　简要介绍目前现场总线技术标准的形成过程。
1.4　简要介绍通信系统的通信指标。
1.5　数据传输方式有哪些？选取其中两种做简要介绍。
1.6　网络传输的介质访问控制方式有哪些？请说明其工作原理。
1.7　OSI 参考模型中物理层、数据链路层有哪些特性？
1.8　简要说明 CAN 总线发展历程。
1.9　简要说明 CAN 总线的数据通信方式。

第 2 章　CAN 总线技术及其协议规范

控制器局域网络（CAN）主要用于各种过程（设备）监测及控制。CAN 最初是由德国的 Bosch 公司为汽车的监测与控制设计的，但由于 CAN 本身的突出特点，其应用领域目前已不再局限于汽车行业，而向过程工业、机械工业、机器人、数控机床、医疗器械和武器装备等领域发展。由于其高性能、高可靠性及独特的设计，CAN 已成为工业数据通信的主流技术之一，并形成了国际标准（ISO 11898）。本章将主要介绍 CAN 总线的技术及其协议规范。

本章要点

- CAN 总线的系统组成；
- CAN 总线通信参考模型；
- CAN 总线的报文传送和帧结构；
- CAN 总线的报文编码、滤波和校验，错误类型和界定；
- CAN 总线的位定时与同步技术。

2.1　CAN 总线技术及其协议规范概述

由于 CAN 被越来越多的领域采用和推广，从而要求不同领域通信报文标准化。为此，Philips Semiconductors 于 1991 年制定并发布了 CAN 总线技术规范（Version2.0），分为两部分：2.0A 给出了曾在 1.2 版规范中定义的 CAN 报文格式，即标准格式；2.0B 给出了标准和扩展的两种报文格式。1993 年 11 月，ISO 正式颁布了道路交通运载工具–数字信息交换–高速通信控制器局域网（CAN）国际标准（ISO 11898），为控制器局域网 CAN 的标准化、规范化推广奠定了基础。

CAN 由于通信速率高、可靠性好以及价格低等特点，使其应用领域已不再局限于传统汽车行业，而向铁路、武器装备和一般工业自动化领域发展。

CAN 总线技术规范的目的是为了在任何两个 CAN 节点之间建立兼容性。目前各个公司生产的 CAN 控制器都支持 CAN2.0B 版本协议规范，且具有向下兼容性。本章主要介绍 CAN2.0B 协议规范。

2.2　CAN 总线的系统构成

CAN 总线是一种串行多主站控制器局域网总线，也是一种有效支持分布式控制或实时控制的串行通信网络。CAN 总线的通信介质可以是双绞线、同轴电缆或光导纤维，通信速率可达 1Mbit/s（电缆长度 40m），通信距离可达 10km（通信速率为 5kbit/s）。CAN 总线的系统组成、拓扑结构和传输介质如下。

2.2.1　CAN 总线的系统组成

CAN 在硬件成本上很具优势，从硬件芯片上来说，智能节点要收发信息需要一个 CAN 控制器和一个 CAN 收发器。经过 20 多年的发展，CAN 已经获得了国际上各大半导体制造商的大力支持，据 CAN 最主要的推广组织 CIA（自动化 CAN）统计，目前已经有 20 余种 CAN 控制器和收发器可供选择，片内集成 CAN 控制器的单片机更多达 100 余种。CAN 在开发成本上的优势也很明显。目前，从广泛应用的 8bit/16bit 单片机，到 DSP 和 32bit 的 PowerPC、ARM 等嵌入式处理器，均在芯片内部含有 CAN 总线硬件接口单元。因此，从硬件角度来看，CAN 具备其他现场总线无法比拟的高集成化优势和广泛的市场支持基础。CAN 总线的开发平台也比较简单，用户如果选择普通单片机加上 CAN 控制器进行开发，CAN 的开发平台和普通单片机的开发平台完全相同。如果选择带有片内 CAN 控制器的单片机进行开发，则只要换用支持该单片机的仿真器就可以了，其他开发设备完全相同。开发 CAN 也需要相应的驱动程序。用户可以自行根据选择的 CAN 控制器开发驱动程序。

典型的 CAN 总线结构图如图 2-1 所示，其中使用微处理器负责 CAN 总线数据处理，完成收发启动等特定功能。CAN 控制器（如 SJA1000）扮演实现网络协议的角色， 它提供了微处理器的物理线路的接口， 进行数据的发送和接收。CAN 总线驱动器（PCA82C250）提供了 CAN 控制器与物理总线之间的接口，实现对 CAN 总线的差动发送和接收功能，是影响系统网络性能的关键因素之一。

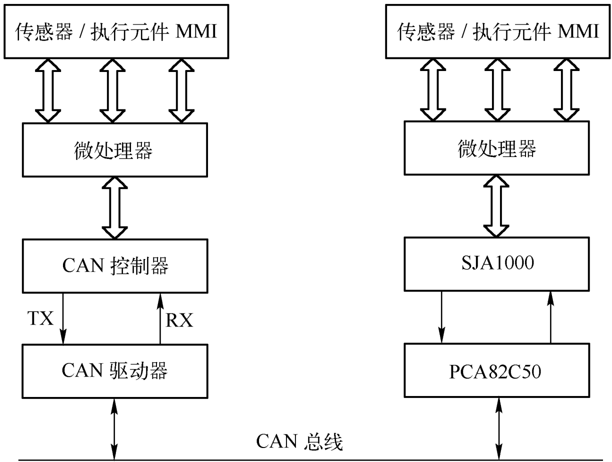


图 2-1　CAN 总线系统构成示意图

2.2.2　CAN 总线的拓扑结构

CAN 总线是一种分布式的控制总线，总线上的每一个节点一般来说都比较简单，通过 CAN 总线将各节点连接只需要较少的线缆，可靠性也较高。

1．总线结构拓扑

ISO 11898 定义了一个总线结构的拓扑，采用干线和支线的连接方式：干线的两个终端都端接一个 120Ω 的终端电阻；节点通过没有端接电阻的支线连接到总线，CAN 总线网络结构

如图 2-2 所示。

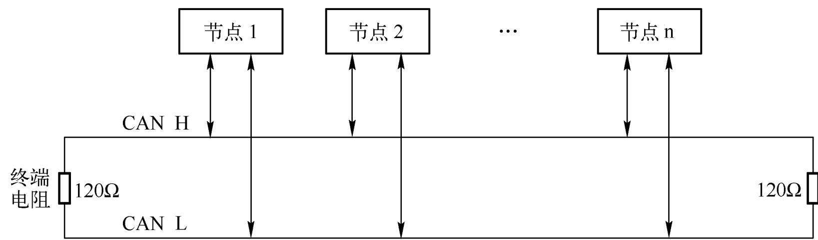


图 2-2　CAN 总线网络结构示意图

在实际应用中，可通过 CAN 中继器将分支网络连接到干线网络上，每条分支网络都符合 ISO 11898 标准，这样可以扩大 CAN 总线通信距离，增加 CAN 总线工作节点的数量，如图 2-3 所示。

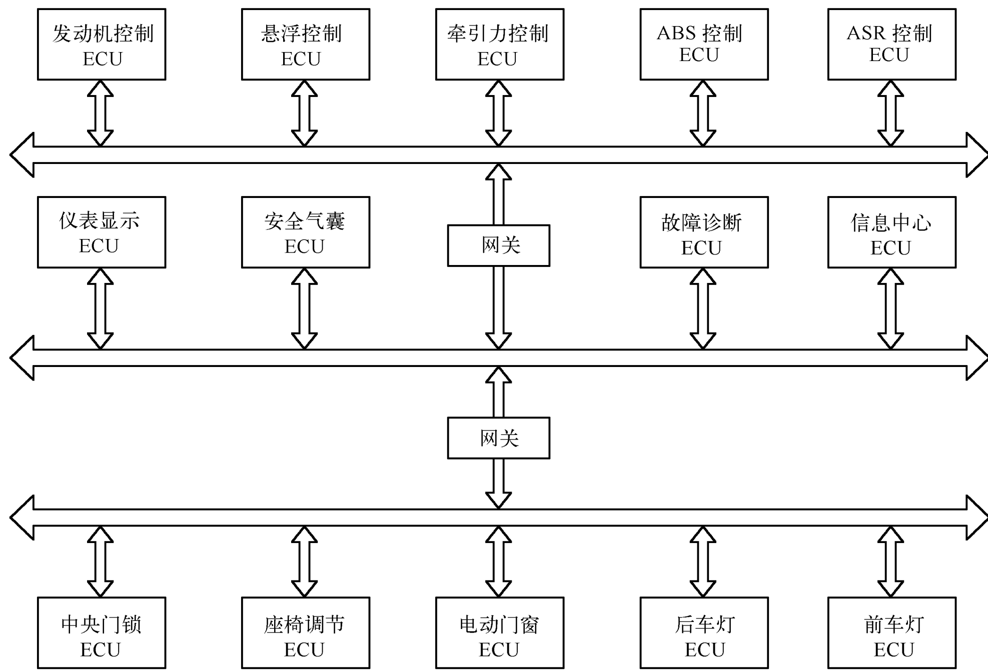


图 2-3　CAN 总线扩展网络示意图

2．CAN 总线通信距离

ISO 11898 规定了 CAN 总线的干线与支线的参数，见表 2-1。CAN 总线最大通信距离与其位速率有关，具体关系见表 2-2，具体说明见第 7.3 节。

表 2-1　CAN 总线的干线与支线的参数

CAN 总线位速率	总 线 长 度	支 线 长 度	节 点 距 离
1Mbit/s	最大 40m	最大 0.3m	最大 40m
5kbit/s	最大 10km	最小 6m	最小 10km

表 2-2　CAN 总线最大通信距离与其位速率关系

位速率/（kbit/s）	5	10	20	50	100	125	250	500	1000
最大有效距离/m	10 000	6 700	3 300	1 300	620	530	270	130	40

2.2.3　CAN 总线的传输介质

CAN 总线可使用多种传输介质，常用的如双绞线、同轴电缆、光纤等，同一段 CAN 总线网络要采用相同的传输介质。基于双绞线的 CAN 总线分布系统已得到广泛应用，ISO 11898 推荐电缆及参数见表 2-3。其主要特点如下：

① 双绞线采用抗干扰的差分信号传输方式；

② 技术上易实现、造价低；

③ 对环境电磁辐射有一定抑制能力；

④ 使用非屏蔽双绞线时，只需要 2 根线缆作为差分信号线传输；

⑤ 使用屏蔽双绞线时，除需要 2 根差分信号线的连接以外，还要注意在同一网络段中的屏蔽层单点接地问题。

表 2-3　ISO11898 推荐电缆及参数

总线长度/m	电　缆		终端电阻/Ω（精度 1%）	最大位速率/（Mbit/s）
	直流电阻/（mΩ/m）	导线截面积/ mm^2		
0～40	70	0.25～0.34 AWG23,AWG22	124	1（40m）
40～300	<60	0.34～0.60 AWG20,AWG22	127	1（100m）
300～600	<40	0.50～0.60 AWG20	127	1（500m）
600～1000	<26	0.75～0.80 AWG18	127	1（1000m）

在使用双绞线搭建 CAN 总线网络时，应注意以下问题：

① 干线两端必须各有一个约 120Ω的终端电阻；

② 支线必须尽可能地短，必要时可以采用手拉手的连接方案，即干线尽可能地接近每个节点；

③ CAN 总线网络线不要布置在干扰源附近；

④ 在外界干扰较大的场所，CAN 总线可采用带屏蔽层的双绞线；

⑤ 使用的电缆的电阻必须足够小，以避免线路压降过大；

⑥ 波特率的选择取决于传输线的延时，CAN 总线的通信距离随着波特率的减小而增加。

2.3　CAN 总线通信参考模型

根据 ISO/OSI 参考模型，CAN 被分为物理层（Physical Layer）和数据链路层（Data Link

Layer，包括媒体访问控制子层 MAC 和逻辑链路控制子层 LLC）。CAN 的 ISO/OSI 参考模型的层结构如图 2-4 所示。

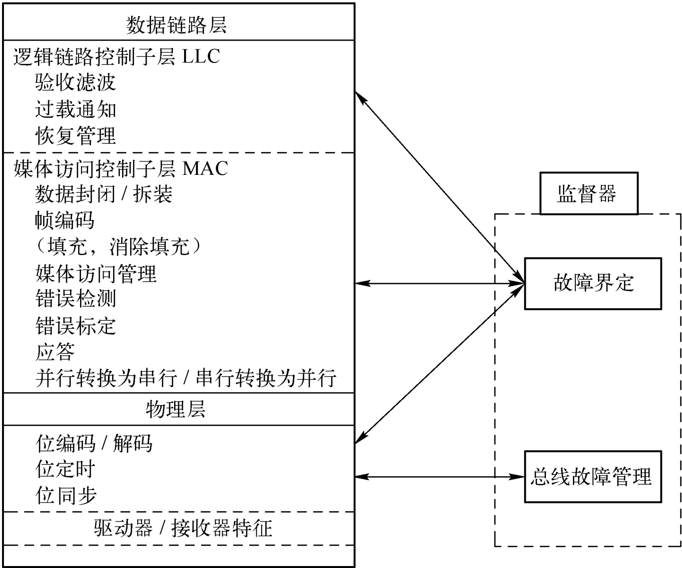


图 2-4　CAN 的 ISO/OSI 参考模型的层结构

具体说明如下：

物理层（Physical Layer）定义信号的实际传输方式，涉及位编码/解码、位定时和位同步等，在同一网络内，要实现不同节点间的数据通信，所有节点的物理层必须一致，CAN2.0 技术规范没有定义物理层的驱动器/接收器特征，以便允许根据它们的应用，对发送媒体和信号电平进行优化。

数据链路层（Data Link Layer）包含媒体访问（MAC）和逻辑链路控制（LLC）两个子层：

1）MAC 子层是 CAN 协议的核心。它把接收到的报文提供给 LLC 子层，并接收来自 LLC 子层的报文。MAC 子层主要规定了传输规则，即负责控制帧的结构、执行仲裁、应答、错误检测、错误标定以及故障界定等。总线何时发送新报文以及何时开始接收报文，均由 MAC 子层来确定，另外位定时也是由 MAC 子层的一部分来确定。

2）LLC 子层涉及验收滤波、过载通知以及恢复管理。

2.4　CAN 总线报文的传送

在进行数据传送时，发出报文的节点为该报文的发送器，该节点在总线空闲或丢失仲裁前恒为发送器。如果一个节点不是报文的发送器，并且总线不处于空闲状态，则该节点为接收器。对于报文的接收器和发送器，报文的实际有效时刻是不同的。对于发送器而言，如果直到帧结束末尾一直未出错，则对于发送器报文才有效。

构成一帧的帧起始、仲裁场、控制场、数据场和 CRC 序列均借助位填充规则进行编码。

当发送器在发送的位流中检测到 5bit 连续的相同数值时，将自动在实际发送的位流中插入一个补码位。而数据帧和远程帧的其余位场则采用固定格式，不进行填充，出错帧和超载帧同样是固定格式。报文中的位流是按照非归零（NZR）码方法编码的，这意味着一个完整的位电平要么是显性，要么是隐性。在隐性状态下，CAN 总线的 V_{CANH} 和 V_{CANL} 被固定于平均电压电平，V_{diff} 近似为零。而在显性状态下，V_{diff} 为大于最小阀值的差分电压，如图 2-5 所示。

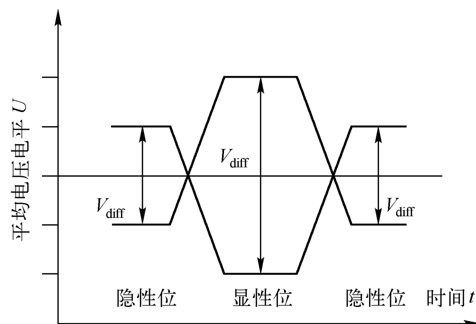


图 2-5　总线上的位电平表示

CAN 协议定义了两种逻辑状态位，且约定逻辑 0 为显位，逻辑 1 为隐位。显位在与隐位争夺总线的过程中将获胜，也就是如果有多个站点在发送信号，只要有一个站点发送了显位，总线就表现为显位。站点在发送的同时也对总线进行检测，以判断总线状态是否就是刚刚发出的信息。在总线仲裁过程中，一个优先级较低的报文在某一时刻会发送一个隐位但检测回来的却是显位，此时该站点被仲裁为发送权取消，它立刻停止发送报文的工作。优先级较高的报文则继续发送。在冲突仲裁中被取消发送权的站点将等待总线的下一个空闲期，并自动再次尝试发送。在显性位期间，显性状态改写隐性状态并发送。

2.5　CAN 总线报文的帧结构

CAN 总线报文有以下 4 种帧类型：

1）数据帧：数据帧将数据从发送器传输到接收器。

2）远程帧：总线节点发出远程帧，请求发送具有同一识别符的数据帧。

3）错误帧：任何节点检测到总线错误就发出错误帧。

4）超载帧：超载帧用以在先行的和后续的数据帧（或远程帧）之间提供一附加的延时。

CAN 总线的帧结构（适用 CAN 协议的 1.2 版和 2.0 版）如下：

1）CAN1.2 定义的报文为标准格式。

2）CAN2.0A 定义的报文为标准格式。

3）CAN2.0B 定义了标准和扩展的两种报文格式，只有数据帧和远程帧可以使用标准帧和扩展帧两种格式，其主要区别在于标识符的长度，具有 11bit 标识符的帧称为标准帧，而包括 29bit 标识符的帧称为扩展帧。

1．数据帧

数据帧由 7 个不同的位场组成：帧起始（Start of Frame）、仲裁场（Arbitration Frame）、控制场（Control Frame）、数据场（Data Frame）、CRC 场（CRC Frame）、应答场（ACK Frame）、帧结束（End of Frame）。数据场的长度可以为 0。报文的数据帧一般结构如图 2-6 所示。

（1）帧起始

帧起始（SOF）标志数据帧和远程帧的起始，仅由一个“显性”位组成。只有在总线空闲时才允许节点开始发送（信号）。所有节点必须同步于首先开始发送报文的节点的帧起始前

沿，如图 2-7 所示。

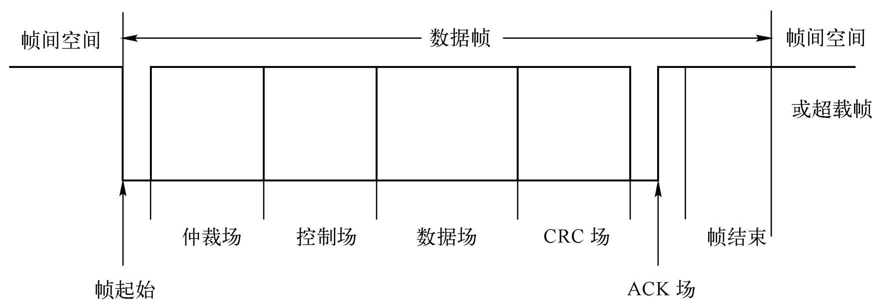


图 2-6　报文的数据帧结构

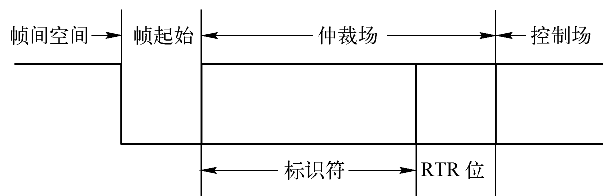


图 2-7　CAN 的帧起始

（2）仲裁场

标准帧的仲裁场由 11bit ID（标识符）和 RTR 位（远程发送请求位）组成，如图 2-8 所示。11bit 标识符按 ID.10 到 ID.0 的顺序发送，RTR 位在数据帧中为显性，在远程帧中为隐性。

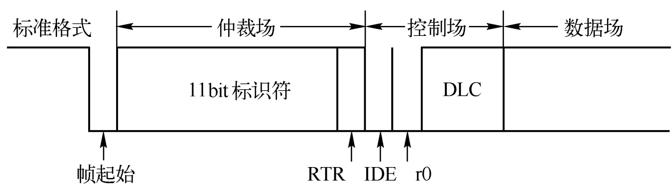


图 2-8　标准格式的仲裁场与控制场

扩展帧的仲裁场由 11bit 基本 ID、SRR 位（替代远程请求位）、IDE 位（标识符扩展位）、18bit 扩展 ID 和 RTR 位组成，如图 2-9 所示。扩展帧的基本 ID 如同标准帧的标识符。其 SRR 是一隐性位，它在相当于标准帧的 RTR 位上被发送，并代替标准帧的 RTR 位。这样，标准帧与扩展帧的冲突通过标准帧优先于扩展帧这一途径得以解决。对于 IDE，在扩展格式中它属于仲裁场，为隐性，在标准格式中它属于控制场，为显性。因此，通过 SRR 和 IDE 位共同确定，是标准帧还是扩展帧，并通过 RTR 位确定是否为远程帧。

（3）控制场

控制场由 6 个位组成，如图 2-10 所示。标准格式的控制场由 IDE 位、保留位 r0 和 DLC（数据长度码）组成，扩展格式里是 r1 和 r0 两个保留位。其保留位发送必须为显性，但接收器对显性和隐性都认可。

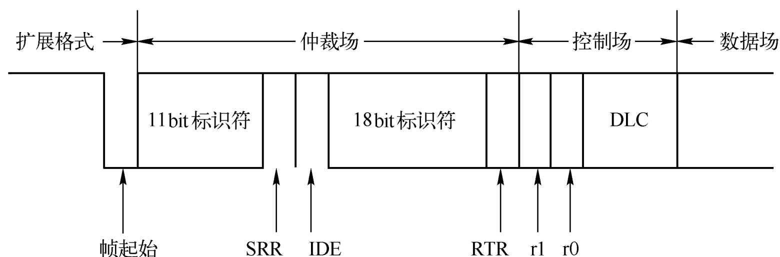


图 2-9 扩展格式的仲裁场与控制场

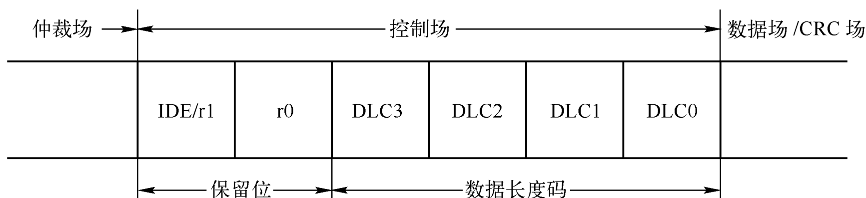


图 2-10 控制场的结构

DLC 指示数据场里的字节数量，其对应关系见表 2-4，其中，d 表示显性，r 表示隐性，数据字节数的范围为 0～8，其他值不允许使用。

表 2-4 数据长度码 DLC 与数据字节数的关系

数据字节的数目	数据长度代码			
	DLC3	DLC2	DLC1	DLC0
0	d	d	d	d
1	d	d	d	r
2	d	d	r	d
3	d	d	r	r
4	d	r	d	d
5	d	r	d	r
6	d	r	r	d
7	d	r	r	r
8	r	d	d	d

（4）数据场

数据场由数据帧里的发送数据组成。它可以为 0～8B，每个字节包含了 8 个位，首先发送最高有效位。

（5）CRC 场

CRC 场包括 CRC 序列和 CRC 界定符，如图 2-11 所示。CRC 序列就是循环冗余码检查序列。在进行 CRC 计算时，被除的多项式由无填充的位流给定，组成这些位流的成分包括帧起始场、仲裁场、控制场和数据场，除数多项式为

$$x^{15}+x^{14}+x^{10}+x^{8}+x^{7}+x^{4}+x^{3}+1$$

CRC 界定符是一个单独的隐性位。

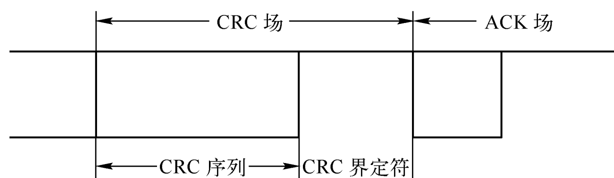


图 2-11　CRC 场的结构

（6）应答场

应答场的长度为 2 个位，由 ACK 间隙和 ACK 界定符组成，如图 2-12 所示。在应答场里，发送站发送两个隐性位。所有接收到匹配 CRC 序列的站，在 ACK 间隙期间，用一显性位写入发送器的隐性位来做出回答。

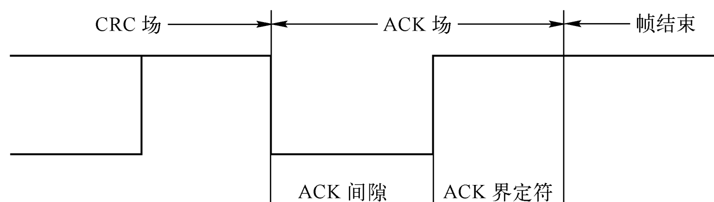


图 2-12　应答场

ACK 界定符是一个隐性位。ACK 间隙被 CRC 界定符和 ACK 界定符两个隐性位所包围。

（7）帧结束场

每一个数据帧和远程帧均由一标志序列界定，该标志序列由 7 个隐性位组成。

2．远程帧

作为接收器的站点，可以通过向相应的数据源站点发送远程帧激活该源站点，让该源站点把数据发送给接收器。远程帧由 6 个位场组成：帧起始场、仲裁场、控制场、CRC 场、应答场、帧结束场，如图 2-13 所示。

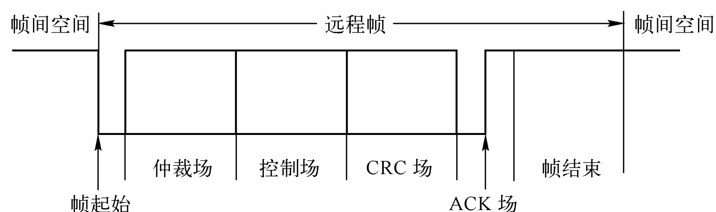


图 2-13　远程帧结构

与数据帧不同之处在于，远程帧的 RTR 位是隐位，没有数据场，数据长度码的值可以是请求的数据的长度值。

3．错误帧

错误帧由错误标志叠加场和错误帧界定符组成，如图 2-14 所示。

错误标志分为“错误激活”标志和“错误认可”标志。“错误激活”标志由 6 个连续的显位组成，“错误认可”标志由 6 个连续的隐位组成。“错误激活”标志由“错误激活”站点（出

错较少的站点）发出，“错误认可”标志由“错误认可”站点（出错较多的站点）发出。检测到出错条件的“错误激活”站点发送“错误激活”标志来指示错误，该标志的格式破坏了从帧起始场到 CRC 界定符的位填充规则，或者破坏了应答场或帧结束场的固定格式，因而，其他站点将检测到错误条件并发送错误标志。这样，在总线上被监视到的显位序列是由各个站点单独发送的错误标志叠加而形成的，该序列的长度在 6～12bit 之间。

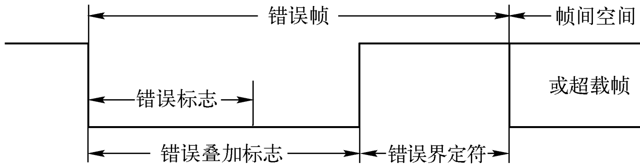


图 2-14　错误帧结构

检测到错误条件的站点，在发送完错误标志以后，就向总线发送隐位并监测总线，直到检测到 1 个隐位为止，然后它继续向总线发送 7 个隐位，这共 8 个隐位称为错误界定符。检测到出错条件的站点，从检测到第 1 个隐位开始，检测到连续的 6 个隐位时，就对本次出错进行了认可。

4．超载帧

有两种超载条件引发超载帧的发送，其一是接收器内部对于下一数据帧或远程帧需要一定延时，其二是在间歇场中检测到显性位。前者引发的超载帧将在下一预期间歇场的第 1 个位上发送，而后者引发的超载帧在检测到显性位之后立即发送。

超载帧包括超载标志场和超载界定符，如图 2-15 所示。

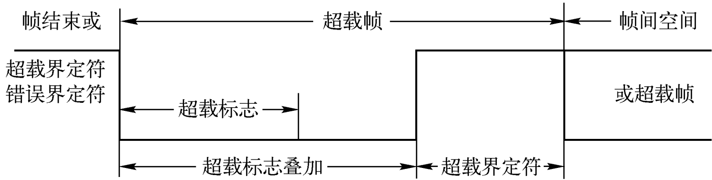


图 2-15　超载帧结构

超载帧与错误帧的形式相同。超载标志由 6 个显性位组成，其格式破坏了间歇场的固定格式，因此，所有其他站点都检测到超载条件并发出超载标志。发完超载标志后，站点就一直发送隐性位并监视总线，直到检测到 1 个隐性位，然后它继续向总线发送 7 个隐性位，这 8 个隐性位称为超载界定符。

5．帧间空间

无论先行帧是何种类型，数据帧或远程帧与先行帧都通过帧间空间来分开。超载帧与错误帧之前没有帧间空间，多个超载帧之间也可没有帧间空间。

普通的帧间空间由间歇场和总线空闲场组成，如图 2-16 所示。间歇场为 3 个隐性位。其实现方法是，数据帧或远程帧在检测到帧结束场后，在发送数据之前，要等待 3 个位时间。在间歇场，所有站点均不允许发送数据帧或远程帧，如果哪个站点发送了，就会被其他的站点指出。总线空闲场的时间是任意的，在此期间，所有等待发送报文的站就会访问总线，在

总线空闲场上检测到的显性位被解释为帧起始场。

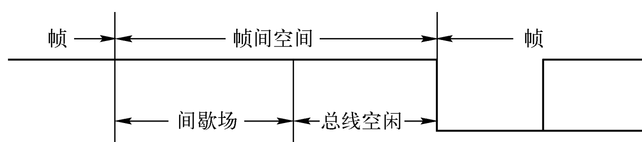


图 2-16　普通的帧间空间

如果某发送器为“错误认可”站点，则其帧空间在间歇场和总线空闲场之间还要插入一个暂停发送场，如图 2-17 所示。暂停发送场是 8 个隐性位。

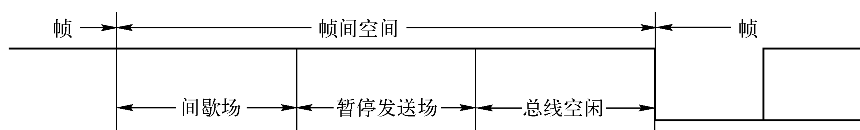


图 2-17 “错误认可”站点发送前的帧间空间

2.6　CAN 总线报文的编码、滤波和校验

1．报文编码

编码即位流编码（Bit Stream Coding），它的规定如下：

1）帧的帧起始、仲裁场、控制场、数据场以及 CRC 序列，均通过位填充的方法编码。无论何时，发送器只要检测到位流里有 5 个连续相同值的位，便自动在位流里插入一补充位。

2）数据帧或远程帧（CRC 界定符、应答场和帧结束）的剩余位场形式固定，不填充。错误帧和过载帧的形式也固定，但并不通过位填充的方法进行编码。

3）报文里的位流根据“不归零”（NRZ）的方法来编码，即在整个位时间里，位的电平要么为“显性”，要么为“隐性”。

2．报文滤波

报文滤波取决于整个识别符。为了实现报文滤波的灵活控制，通过初始化验收屏蔽寄存器，允许在报文滤波中将任何的识别符位设置为“不考虑”位。

在使用屏蔽寄存器时，它的每一个位都是可编程的，即它们能够被设置成允许或禁止报文滤波。屏蔽寄存器的长度可以包含整个识别符，也可以包含部分的识别符。

3．报文校验

校验报文有效的时间点，对发送器与接收器来说各不相同。

（1）发送器（Transmitter）

如果直到帧的末尾位均没有错误，则报文对于发送器有效。如果报文出错，则报文根据优先权自动重发。为了能够和其他报文竞争总线，重新传输必须在总线空闲时启动。

（2）接收器（Receiver）

如果直到最后的位（除了帧末尾）均没有错误，则报文对于接收器有效。帧末尾最后的位被置于“不重要”状态，如果是一个“显性”电平也不会引起格式错误。

2.7 CAN 总线报文的优先级确定问题

2.7.1 CAN 总线的仲裁过程

CAN 总线上的节点没有主从之分，所有的节点级别都一样，属于多主掌控方式，都可以作为发送节点，也可以作为接收节点，只要总线空闲，有数据要发送的节点就会往总线上发送数据，在发送数据时，发送节点并不会指定由哪个节点接收，而是由接收节点自己过滤/选择是否接收该数据。

如果两个或多个节点同时发送数据，如何进行冲突仲裁？支配位一定会在和顺从位的判别过程中获胜，换句话说，报文标记区（报文仲裁专用区域）的值越小，其优先级就越高。

假定有两个节点在同一时刻发送一个报文，每个节点都会监测总线以便了解欲发送的信息状态是否确实出现在总线上。一个优先级较低的报文在某一时刻会发送一个“顺从位 1”但是检测回来的却是“支配位”。此时，这个节点被仲裁为发送权取消，立刻停止发送报文的工作。优先级较高的报文继续发送直到完整的报文发送完毕。在刚才冲突仲裁中被取消发送权的节点将等待总线的下一个空闲期并自动地再次尝试发送。举个例子，如果两个节点同时发送数据如下：

节点 1：0000000**0**111 10111011101111111……

节点 2：00000001111 01011111101001010……

发送到第 8bit 时，节点 2 判断出其他节点在发送高优先级的报文，节点 2 自动停止发送，等待总线空闲。

通过以上的逐位仲裁方法来使有最高优先权的报文优先发送，在 CAN 总线报文的帧结构介绍中，其上发送的每一条报文都具有唯一的一个 11bit（标准帧）或 29bit（扩展帧）数字的 ID。ID 号越小，则该报文拥有越高的优先权。因此，一个为全 0 标识符的报文具有总线上的最高级优先权。

在总线空闲时，最先开始发送消息的节点获得发送权，多个节点同时发送时，各发送节点从仲裁段的第一位开始进行仲裁，连续输出显性电平最多的节点可继续发送。

2.7.2 数据帧和远程帧的优先级

对于 CAN 技术规范，标准帧的标识符的长度为 11bit（扩展帧为 29bit），这些位按照从高位到低位的顺序发送，最低位为 ID.0。第 12bit（扩展帧为第 30bit，即 RTR 位）在数据帧中必须为显性，而在远程帧中必须为隐性。具有相同 ID 的数据帧和远程帧在总线上竞争时，仲裁段的最后一位（RTR）为显性位的数据帧具有优先权，可继续发送，而远程帧对应位为隐性位，优先级低。

2.7.3 标准格式和扩展格式的优先级

以具有相同 ID 的两种格式的数据帧和远程帧为例，优先级从高到低分别排列为标准格式的数据帧、标准格式的远程帧、扩展格式的数据帧、扩展格式的远程帧。扩展格式的数据帧

和远程帧在总线上竞争时，如果其他节点同时发送标准格式的数据帧，在发送到第 12bit（RTR 位）时，标准格式的数据帧为对应显性位，具有优先权，可继续发送，而扩展帧的第 12bit 为 SRR 位为隐性，优先级低；当其他节点发送的是标准格式的远程帧时，在发送的第 12bit 均为隐性位，但到第 13bit（IDE 位）时，标准格式的远程帧为显性位，具有优先权，可继续发送，而扩展格式的数据帧或远程帧的第 13bit（IDE 位）为隐性位，优先级低。

2.8 CAN 总线错误处理

1．错误类型

CAN 总线共有以下 5 种不同的错误类型（这 5 种错误不会相互排斥）。

（1）位错误（Bit Error）

节点在发送位的同时也对总线进行监视。如果所发送的位值与所监视的位值不相合，则在此位时间里检测到一个位错误。但是在仲裁场（Arbitration Field）的填充位流期间或应答间隙（ACK Slot）发送一“隐性”位的情况是例外的。此时，当监视到一“显性”位时，不会发出位错误。当发送器发送一个“认可错误”标志但检测到“显性”位时，也不视为位错误。

（2）填充错误（Stuff Error）

如果在使用位填充法进行编码的信息中，出现了 6 个连续相同的位电平时，将检测到一个填充错误。

（3）CRC 错误（CRC Error）

CRC 序列包括发送器的 CRC 计算结果。接收器计算 CRC 的方法与发送器相同。如果计算结果与接收到 CRC 序列的结果不相符，则检测到一个 CRC 错误。

（4）格式错误（Form Error）

当一个固定形式的位场含有 1 个或多个非法位，则检测到一个格式错误。（备注：接收器的帧末尾最后一位期间的显性位不被当作帧错误）

（5）应答错误（Acknowledgment Error）

只要在应答间隙期间所监视的位不为“显性”，则发送器会检测到一个应答错误。

2．错误标志

检测到错误条件的节点通过发送错误标志指示错误。对于“错误激活”的节点，错误信息为“激活错误”标志；对于“错误认可”的节点，错误信息为“认可错误”标志。节点检测到无论是位错误、填充错误、格式错误，还是应答错误，这个节点会在下一位时发出错误标志信息。

只要检测到的错误的条件是 CRC 错误，错误标志的发送开始于 ACK 界定符之后的位（除非其他错误条件引起的错误标志已经开始）。

2.9 CAN 总线故障界定

CAN 总线出错的原因可能是总线扰动，也可能是节点出现不可恢复的故障，在 CAN 总线技术规范中详细定义了用于故障界定的故障状态及转换规则。

2.9.1 故障界定方法

一个 CAN 总线节点出错后可能处于以下 3 种状态中的一种，包括错误激活状态、错误认可状态和总线关闭状态。

1．CAN 的 3 种故障状态

（1）错误激活（Error Active）

“错误激活”的节点可以正常地参与总线通信，并在错误被检测到时发出“激活错误”标志。一些教材也有译为“主动错误状态”。

（2）错误认可（Error Passive）

“错误认可”节点不允许发送“激活错误”标志。当“错误认可”节点参与总线通信时，在错误被检测到时只发出“认可错误”标志。而且，发送之后，“错误认可”节点将在启动下一个发送之前处于等待状态。一些教材也有译为“被动错误状态”。

（3）总线关闭（Bus Off）

“总线关闭”的节点不允许对总线产生任何的影响（比如，关闭输出驱动器）。

2．CAN 的 2 种故障计数器

在每一总线节点使用两种计数器以便故障界定，包括：

① 发送错误计数；

② 接收错误计数。

2.9.2 错误计数规则

这些故障计数器按以下规则改变（注意，在给定的报文发送期间，可能要用到的规则不只一个）。

1）当接收器检测到一个错误，接收错误计数器值就加 1。在发送“认可错误”标志或过载标志期间，所检测到的错误为位错误时，接收错误计数器值不加 1。

2）当错误标志发送以后，接收器检测到的第一个位为“显性”时，接收错误计数器值加 8。

3）当发送器发送一错误标志时，发送错误计数器值加 8。在以下例外情况 1 和例外情况 2 发生时，发送错误计数器值不改变。

例外情况 1：发送器为“错误认可”，并检测到应答错误（在应答错误中检测不到显性位），而且在发送“认可错误”标志时也检测不到“显性”位。

例外情况 2：发送器由于在仲裁期间发生填充错误，此填充位应该为隐性位，但却检测出显性位，发送器送出错误标志。

4）发送“激活错误”标志或过载标志时，如果发送器检测到位错误，则发送错误计数器值加 8。

5）发送“激活错误”标志或过载标志时，如果接收器检测到位错误，则接收错误计数器值加 8。

6）在发送“激活错误”标志、“认可错误”标志或过载标志以后，任何节点最多容许 7 个连续的“显性”位。在以下 3 种情况，每一发送器将它们的发送错误计数值加 8，同时每一接收器的接收错误计数值加 8。

① 当检测到第 14 个连续的“显性”位后；

② 在检测到第 8 个连续的“显性”位跟随在“认可错误”标志后；

③ 在每一个附加的 8 个连续“显性”位序列后。

7）报文成功传送后（得到 ACK 及直到帧末尾结束没有错误），发送错误计数器值减 1，除非已经是 0。

8）报文成功接收后（直到应答间隙接收没有错误，并成功地发送了 ACK 位），如果接收错误计数值介于 1～127 之间，接收错误计数器值减 1。如果接收错误计数器值是 0，则它保持 0；如果大于 127，则它会设置一个介于 119～127 之间的值。

9）当发送错误计数器值等于或超过 128 时，或当接收错误计数器值等于或超过 128 时，节点为“错误认可”。

10）当发送错误计数器值大于或等于 256 时，节点为“总线关闭”。

11）当发送错误计数器值和接收错误计数器值都小于或等于 127 时，“错误认可”节点重新变为“错误激活”节点。

12）在总线监视到 128 次出现 11 个连续“隐性”位之后，“总线关闭”的节点可以变成“错误激活”节点，它的两个错误计数值也被置为 0。

2.10　CAN 总线的位定时

CAN 总线规范中有关位定时的论述包括以下内容。

1．标称位速率

标称位速率为一理想的发送器在没有重新同步的情况下每秒发送的位数量。

2．标称位时间

$$标称位时间 = 1/标称位速率$$

可以把标称位时间划分成几个不重叠时间的片段，它们是同步段（SYNC-SEG）、传播段（PROP-SEG）、相位缓冲段 1（PHASE-SEG1）、相位缓冲段 2（PHASE-SEG2），如图 2-18 所示。

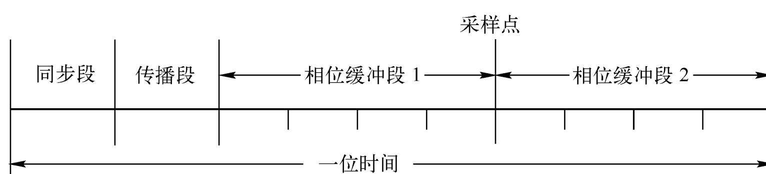


图 2-18　标称位时间的划分

（1）同步段（SYNC-SEG）

位时间的同步段用于同步总线上不同的节点。这一段内要有一个跳变沿。

（2）传播段（PROP-SEG）

传播段用于补偿网络内的物理延时时间。它是信号在总线上传播的时间、输入比较器延时和输出驱动器延时总和的两倍。

（3）相位缓冲段 1、相位缓冲段 2（PHASE-SEG1、PHASE-SEG2）

相位缓冲段用于补偿边沿阶段的误差。这两个段可以通过重新同步加长或缩短。

（4）采样点（Sample Point）

采样点是读总线电平并解释各位的值的一个时间点。采样点位于相位缓冲段 1（PHASE-SEG1）的结尾。

3．信息处理时间

信息处理时间是一个以采样点作为起始的时间段。采样点用于计算后续位的位电平。

4．时间份额

时间份额是派生于振荡器周期的固定时间单元。存在有一个可编程的预比例因子，其数值范围为 1～32 的整数。以最小时间份额为起点，时间份额的长度为

时间份额（Time Quantum）= m×最小时间份额（Minimum Time Quantum）

其中，m 为预比例因子。

5．时间段的长度

① 同步段（SYNC-SEG）为 1 个时间份额；

② 传播段（PROP-SEG）的长度可设置为 1，2，…，8 个时间份额；

③ 相位缓冲段 1（PHASE-SEG1）的长度可设置为 1，2，…，8 个时间份额；

④ 相位缓冲段 2（PHSAE-SEG2）的长度为相位缓冲段 1（PHASE-SEG1）和信息处理时间（Information Processing Time）之间的最大值。

信息处理时间少于或等于 2 个时间份额。

一个位时间总的时间份额值可以设置在 8～25 之间，位时间的定义如图 2-19 所示。

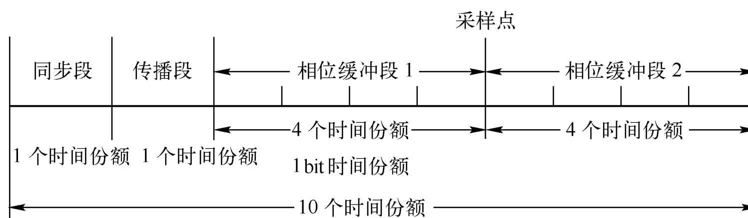


图 2-19　位时间各组成部分

2.11　CAN 总线的位同步

在位定时中，还有一个重要概念就是同步，下面分别进行介绍。

（1）硬同步（Hard Synchronization）

硬同步后，内部的位时间从同步段重新开始。因此，硬同步强迫由于硬同步引起的跳变沿处于重新开始的位时间同步段之内。

（2）重新同步跳转宽度（Resynchronization Jump Width）

重新同步的结果使相位缓冲段 1 增长，或使相位缓冲段 2 缩短。相位缓冲段加长或缩短的数量有一个上限，此上限由重新同步跳转宽度给定。重新同步跳转宽度应设置于 1 和最小值之间（此最小值为 4，PHASE-SEG1）。

可以从一位值转换到另一位值的过渡过程得到时钟信息。这里有一个属性，即：只有后

续位的一固定最大数值才具有相同的数值。这个属性使总线单元在帧期间重新同步于位流成为可能。可用于重新同步的两个过渡过程之间的最大的长度为 29 个位时间。

（3）边沿的相位误差（Phase Error of an Edge）

一个边沿的相位误差由相关于同步段的边沿的位置给出，以时间份额度量。相位误差定义如下：

如果沿处于同步段里（SYNC-SEG），e=0。

如果沿位于采集点（Sample Point）之前，e>0。

如果沿处于前一个位的采集点（Sample Point）之后，e<0。

（4）重新同步（Resynchronization）

当引起重新同步沿的相位误差的幅值小于或等于重新同步跳转宽度的设定值时，重新同步和硬件同步的作用相同。当相位错误的量级大于重新同步跳转宽度时：

1）如果相位误差为正，则相位缓冲段 1 被增长。增长的范围为与重新同步跳转宽度相等的值。

2）如果相位误差为负，则相位缓冲段 2 被缩短。缩短的范围为与重新同步跳转宽度相等的值。

（5）同步的原则（Synchronization Rules）

硬同步和重新同步是同步的两种形式，遵循以下规则：

1）在一个位时间里只允许一个同步。

2）仅当采集点之前探测到的值与紧跟沿之后的总线值不相符合时，才把沿用作于同步。

3）总线空闲期间，无论何时，只要有一"隐性"位转变到"显性"位的边沿，硬同步都会被执行。

4）符合规则 1 和规则 2 的所有从"隐性"位转化为"显性"位的边沿可以用作为重新同步。有一例外情况，即：当发送一显性位的节点不执行重新同步而导致一"隐性"位转化为"显性"位边沿，此边沿具有正的相位误差，不能用作于重新同步。

2.12　本章小结

CAN 总线是一种串行多主站控制器局域网总线，是一种有效支持分布式控制或实时控制的通信网络。本章在介绍 CAN 总线组成基础上，介绍了 CAN 总线的通信参考模型，对 CAN 总线协议规定的帧结构、报文编码、报文优先级确定、错误及故障处理、位定时和同步进行了介绍。

思考题与习题

2.1　简要说明 CAN 总线系统的主要构成，各部分的主要作用。

2.2　简要说明 CAN 总线有哪几种不同类型的帧？

2.3　在 CAN 技术规范 2.0B 中存在两种不同的数据帧格式，其主要区别在于标识符的长度，说明其中的不同格式的数据帧由哪些位场组成，每个场各占多少位？

2.4　对于不同格式的数据帧，计算最长数据帧各由多少位组成？

2.5　CAN 总线中存在哪些不同类型的错误？

2.6　任何一个 CAN 节点都有可能是哪 3 种节点状态？

2.7　结合 CAN 总线技术协议，详细说明 CAN 总线的 3 大特点。

2.8　假设 SJA1000 的晶体振荡频率为 16MHz，总线定时寄存器 0 和 1 参数分别为 0x1f 和 0xff，计算 CAN 通信的位速率。

第 3 章　CAN 总线控制器 SJA1000 及其应用

SJA1000 是 Philips 公司的一种新型独立式控制器，它是 PCA82C200 CAN 控制器（只支持 BasicCAN）的替代产品，在原来基础上增加了一种支持 CAN2.0B 协议的新工作模式—PeliCAN。

本章要点：

- SJA1000 的基本模式和扩展模式下的控制寄存器和数据段寄存器；
- 两种模式下的公共寄存器；
- SJA1000 的读写时序；
- 基于 51 系列单片机的 CAN 智能节点的硬软件设计。

3.1　SJA1000 概述

1．SJA1000 的特性

SJA1000 主要有以下特点：

① 管脚和电气参数同 CAN 独立控制器 PCA82C200 兼容；

② 支持 CAN2.0A 和 CAN2.0B 协议；

③ 支持 11bit 和 29bit 的标识符，支持标准帧和扩展帧的报文的收发；

④ 扩展了接收缓冲器（采用 64B 的 FIFO 队列），减少了数据超载的可能性；

⑤ 位速率最高可达 1Mbit/s，最高时钟频率可达 24MHz；

⑥ 默认模式为与 PCA82C200 兼容的 BasicCAN 模式，同时可选支持 CAN2.0B 协议的 PeliCAN 模式；

⑦ 支持 24MHz 时钟频率；

⑧ 可与不同的微处理器接口；

⑨ 提供可编程的 CAN 输出驱动器配置；

⑩ 温度适应范围大（−40～+125℃）。

SJA1000 在上电复位后自动进入基本模式（Basic Mode），该模式向下兼容 PCA82C200，在复位方式下可对 SJA1000 编程，使其进入扩展模式（Peli Mode）。为了有利于读者学习和了解 SJA1000，本章在介绍 SJA1000 内部结构的基础上，按基本模式和扩展模式下的控制寄存器、数据寄存器和两种模式的公共寄存器分别进行介绍。

2．SJA1000 管脚排列

SJA1000 的引脚定义见表 3−1，引脚排列如图 3−1 所示。

表 3-1　SJA1000 的引脚定义

符　　号	引　　脚	说　　明
AD7～AD0	2，1，23～28	多路地址/数据总线
ALE/AS	3	ALE 输入信号（Intel 模式），AS 输入信号（Motorola 模式）
$\overline{CS}$	4	片选信号输入，低电平允许访问 SJA1000
$\overline{RD}$ /E	5	微控制器的 $\overline{RD}$ 信号（Intel 模式）或 E 使能信号（Motorola 模式）
$\overline{WR}$	6	微控制器的 $\overline{WR}$ 信号（Intel 模式）或 RD（ $\overline{WR}$ ）信号（Motorola 模式）
CLKOUT	7	SJA1000 产生的提供给微控制器的时钟输出信号；时钟信号来源于内部振荡器且通过编程驱动；时钟控制寄存器的时钟关闭位可禁止该引脚输出
VSS1	8	接地
XTAL1	9	输入到振荡器放大电路；外部振荡信号由此输入[①]
XTAL2	10	振荡放大电路输出；使用外部振荡信号时做开路输出[①]
MODE	11	模式选择输入：1=Intel 模式；0=Motorola 模式
VDD3	12	输出驱动的 5V 电压源
TX0	13	从 CAN 输出驱动器 0 输出到物理线路上
TX1	14	从 CAN 输出驱动器 1 输出到物理线路上
VSS3	15	输出驱动器接地
$\overline{INT}$	16	中断输出，用于中断微控制器；$\overline{INT}$ 内部中断寄存器任一位置 1 时，$\overline{INT}$ 被激活，低电平有效；$\overline{INT}$ 是漏极开漏输出，且与系统中的其他 $\overline{INT}$ 是线或的；此引脚上的低电平可以把 IC 从睡眠模式中激活
$\overline{RST}$	17	复位输入，用于复位 CAN 接口（低电平有效）；把 $\overline{RST}$ 引脚通过电容连到 VSS，通过电阻连到 VDD 可自动上电复位（例如，C=1μF；R=50kΩ）
VDD2	18	输入比较器的 5V 电压源
RX0，RX1	19，20	物理 CAN 总线到 SJA1000 的输入比较器的输入；显性电平将会唤醒 SJA1000 的睡眠模式；如果 RX1 比 RX0 的电平高，就读显性（控制）电平，反之读隐性电平；如果时钟分频寄存器的 CBP 位被置位，CAN 输入比较器被忽略以减少内部延时（此时连有外部收发电路）；这种情况下只有 RX0 是激活的
VSS2	21	输入比较器的接地端
VDD1	22	逻辑电路的 5V 电压源

注：XTAL1 和 XTAL2 引脚必须通过 15pF 的电容连到 VSS1。

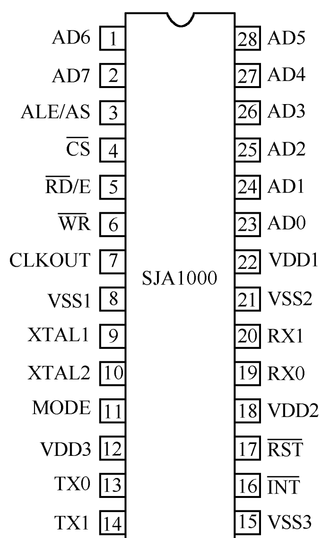


图 3-1　SJA1000 的芯片引脚配置（DIP28）

3.2 SJA1000 的内部结构及其控制模块

SJA1000 的内部结构图如图 3-2 所示，各控制模块说明如下：

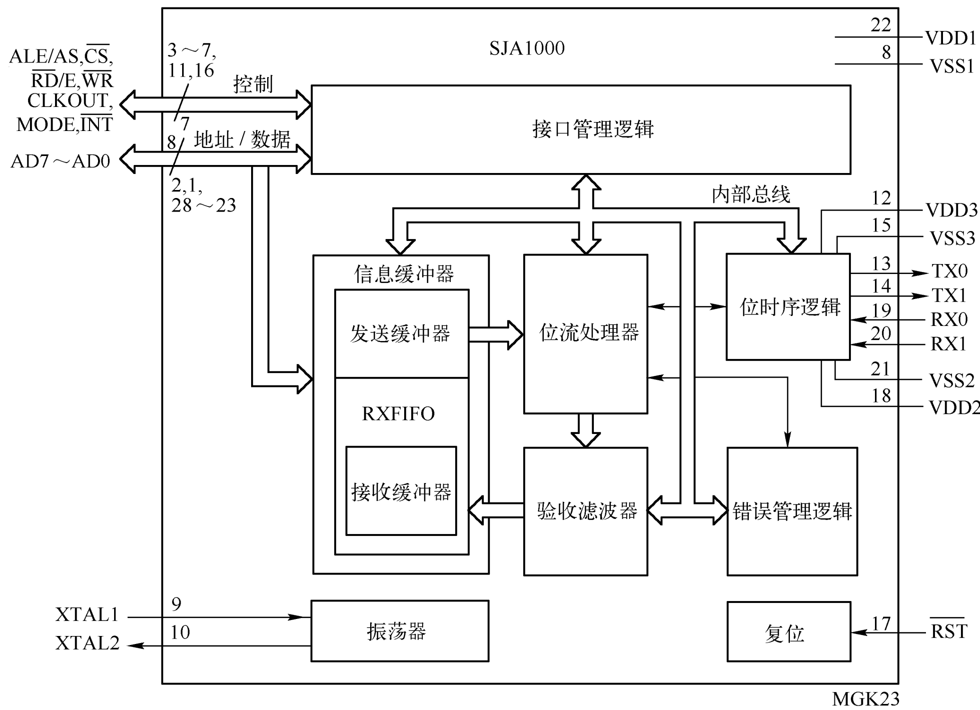


图 3-2　SJA1000 的内部结构方框图

（1）接口管理逻辑（IML）

接口管理逻辑解释来自主控制器 CPU 的命令，控制 CAN 寄存器的寻址，向主控制器提供中断信息和状态信息。

（2）发送缓冲器（TXB）

发送缓冲器是 CPU 和 BSP（位流处理器）之间的接口，能够存储发送到 CAN 网络上的完整信息。缓冲器长 13B，由 CPU 写入、BSP 读出。

（3）接收缓冲器（RXB，RXFIF0）

接收缓冲器是验收滤波器和 CPU 之间的接口，用来储存从 CAN 总线上接收的信息。接收缓冲器（RXB，长 13B）作为接收 FIFO（RXFIFO，长 64B）的一个窗口，可被 CPU 访问。

CPU 在此 FIFO 的支持下，可以在处理信息的时候接收其他信息。

（4）验收滤波器（ACF）

验收滤波器把其中的数据和接收的识别码的内容相比较，以决定是否接收信息。在纯粹的接收测试中，所有的信息都保存在 RXFIFO 中。

（5）位流处理器（BSP）

位流处理器是一个在发送缓冲器、RXFIFO 和 CAN 总线之间控制数据流的程序装置。它还在 CAN 总线上执行错误检测、仲裁、填充和错误处理。

（6）位时序逻辑（BTL）

位时序逻辑监视串口的 CAN 总线和处理与总线有关的位时序。它在信息开头“弱势−支配”的总线传输时同步 CAN 总线位流（硬同步），接收信息时再次同步下一次传送（软同步）。BTL 还提供了可编程的时间段来补偿传播延迟时间、相位转换（例如，由于振荡漂移）、定义采样点和一位时间内的采样次数。

（7）错误管理逻辑（EML）

EML 负责传送层模块的错误管制。它接收 BSP 的出错报告，通知 BSP 和 IML 进行错误统计。

3.3　SJA1000 基本模式下的寄存器

3.3.1　基本模式下的寄存器

SJA1000 在上电复位后自动进入基本模式（Basic Mode），在复位方式下可对 SJA1000 编程，使其进入扩展模式（Peli Mode）。

1．基本模式下的寄存器地址分配

微处理器以存储器映象方式访问 SJA1000 的内部寄存器，以设置 SJA1000 的工作方式和参数。SJA1000 的内部寄存器由控制段和数据段（发送缓冲区和接收缓冲区）组成。在系统初始化时对控制段进行编程，以配置通信参数，微处理器也可通过控制段来控制总线通信。在报文被发送前，微处理器将报文写入发送缓冲区，在成功接收一个报文后，微处理器读接收缓冲区并释放缓冲区。具体的地址分配见表 3−2。

2．SJA1000 的工作状态

SJA1000 有两种工作状态——复位状态和运行状态：复位状态用于 SJA1000 的初始化和总线传输参数的设置；运行状态下，实现对总线数据传输的控制。

（1）复位状态

在以下三种情况下，SJA1000 进入复位状态：

软件复位：通过设置控制寄存器 CR 的 RR 位为 1。

硬件复位：在复位引脚上出现一个低电平脉冲。

BUS_OFF 状态（总线脱离状态）：当发送错误计数器等于或超过 255，总线状态位（BS）置 1，SJA1000 进入 BUS_OFF 状态，控制器自动脱离总线。其间，控制器不接收也不发送总线信息。

（2）运行状态

在 CR 的 RR 位上出现“1-0”的下跳沿时，SJA1000 返回运行状态。可通过检测 RR 来判断 SJA1000 的工作状态。

运行模式与复位模式下，各寄存器的读写状态见表 3−2。

表 3-2　BasicCAN 寄存器地址分配

寄存器地址	寄存器	运行模式		复位模式	
		读	写	读	写
0	控制寄存器 CR	√	√	√	√
1	命令寄存器 CMR	FFH	√	FFH	√
2	状态寄存器 SR	√	×	√	×
3	中断寄存器 IR	√	×	√	×
4	屏蔽码寄存器 ACR	FFH	×	√	√
5	接收屏蔽寄存器 AMR	FFH	×	√	√
6	总线定时寄存器 0 BTR0	FFH	×	√	√
7	总线定时寄存器 1 BTR1	FFH	×	√	√
8	输出控制寄存器 OCR	FFH	×	√	√
9		仅用于测试			
10～19	输出缓冲寄存器 TXB	√	√	FFH	×
20～29	输入缓冲寄存器 RXB	√	√	√	√
30	未用	FFH	×	FFH	×
31	时钟分频寄存器 OCR	√	部分 √	√	√

注：‘√’代表可读或可写；‘×’代表不可读或不可写；‘FFH’代表读取结果。

3.3.2　基本模式下的控制寄存器

以下对基本模式下的控制寄存器进行详细描述。

1．控制寄存器 CR

控制寄存器 CR 用于控制 CAN 控制器 SJA1000 的行为，微控制器将它当作可读写存储器使用。各位的定义见表 3-3。

表 3-3　控制寄存器 CR 各位的定义（CAN 地址 0）

CR.7	CR.6	CR.5	CR.4	CR.3	CR.2	CR.1	CR.0
保留未用			OIE	EIE	TIE	RIE	RR

1）OIE：超载中断允许位，置位时超载将会产生中断，复位时不产生超载中断。

2）EIE：错误中断允许位，置位时发生错误或总线状态改变都将产生中断，复位时无错误中断产生。

3）TIE：发送中断允许位，若置位，当一个报文成功发送或发送缓冲区再次可访问时（如在中止的发送命令后），将会产生中断，复位时无发送中断产生。

4）RIE：接收中断允许位，若置位，当一个报文被无误地接收时将会产生接收中断，复位时无接收中断产生。

5）RR：复位请求位，该位置位后，SJA1000 将会终止当前报文的接收或发送而进入复位工作状态。在硬件复位期间或总线状态为 BUS-OFF 时，复位请求位将被置 1。如果这位可通过软件访问，其值的改变将被反映出来，并在内部时钟的下一个上升沿有效（内部时钟频率是外部晶体振荡频率的 1/2）。在硬件复位期间（外部复位引脚为低电平），微控制器不

能将复位请求位 RR 清零。因此，当微控制器发出 RR 清零命令之后，必须检测此位，以确保 SJA1000 进入了运行模式。复位请求位的改变与内部分频时钟同步，复位请求位的读取反映了同步状态。

2．命令寄存器 CMR

一个命令位在 SJA1000 的传输层内启动一个动作。命令寄存器（CMR）对微控制器而言是只写寄存器，对它进行读取时返回 0FFH。在发送两个命令之间，至少要等待一个内部时钟周期。各位的定义见表 3-4。

表 3-4　命令寄存器（CMR）各位的定义（CAN 地址 1）

CMR.7	CMR.6	CMR.5	CMR.4	CMR.3	CMR.2	CMR.1	CMR.0
保留未用			GTS	CDO	RRB	AT	TR

1）GTS：睡眠命令位，若没有未决中断和总线活动，置“1”将使 SJA1000 进入睡眠状态。进入睡眠状态后，SJA1000 的 CLKOUT 端的输出将持续 15bit 时间，以使主控器通过此信号进入待命状态。下列三种情况下，SJA1000 将被唤醒而进入运行状态：

① GTS 位被复位；

② 出现总线活动；

③ 有未决中断使 $\overline{\text{INT}}$ 端为低。

由于总线活动而被唤醒的 SJA1000 在检测到 11 个连续隐性位（总线空闲标志）前将不能接收总线信息。在复位状态下，该位不可置位。

2）CDO：清除数据超载命令位，置位时有效。

3）RRB：释放接收缓冲器命令位，置位时有效。在 MCU 读取数据后，应该释放接收缓冲器以使 MCU 接收下一批数据。该位和 CDO 可同时有效。

4）AT：中止发送命令位，该命令用于 MCU 请求挂起当前数据发送，例如有更紧急的数据要发送时。但已经在进行的数据发送不会被中止。

5）TR：发送请求位，置位时有效，复位时可取消当前发送。

3．状态寄存器 SR

状态寄存器的内容反映 SJA1000 的状态。它对于 MCU 来说为只读寄存器。各位的定义见表 3-5。

表 3-5　状态寄存器（SR）各位的定义（CAN 地址 2）

SR.7	SR.6	SR.5	SR.4	SR.3	SR.2	SR.1	SR.0
BS	ES	TS	RS	TCS	TBS	DOS	RBS

1）BS：总线状态标志，“1”表示处于总线脱离状态。当发送错误计数器超过 255 的极限时，SJA1000 自动将 BS 置“1”并产生一个错误中断（若允许），同时控制器进入复位工作方式，直到 MCU 清除 BS。

2）ES：错误状态标志，当至少有一个错误计数器达到或超过警告极限时该位被置“1”，若允许错误中断还将产生中断。

3）TS：发送状态标志，“1”表示 SJA1000 正在发送数据。

4）RS：接收状态标志，“1”表示 SJA1000 正在接收数据，若 TS 和 RS 同时为“0”，

则表明总线空闲。

5）TCS：发送完成标志，“1”表示上次的发送已成功完成。

6）TBS：发送缓冲区状态标志，“1”表示发送缓冲区可写。若该位为“0”时，MCU 写发送缓冲区，则写入数据无效且被丢失。

7）DOS：数据超载标志，“1”表示由于 RXFIFO 没有足够的空间，收到的报文丢失。

8）RBS：接收缓冲区状态标志，“1”表示 RXFIFO 中至少有一个报文。当 MCU 读取报文后，应给出释放接收缓冲区的命令，该标志才会清零。若 RXFIFO 中还有未读报文，该位又将被置位。

4．中断寄存器 IR

中断寄存器的内容反映 SJA1000 中断源的状态，它也是只读寄存器。当一个或多个中断源发出中断请求信号，中断寄存器的相应位被置 1（如允许），中断引脚 $\overline{\text{INT}}$ 被触发（低电平有效）。只要微控制器读中断寄存器，中断寄存器被复位，中断引脚 $\overline{\text{INT}}$ 将悬空。由于 SJA1000 只有一个中断输出引脚 $\overline{\text{INT}}$，所以 MCU 应在中断程序中查询该寄存器，以确定是何种中断。各位的定义见表 3-6。

表 3-6　中断寄存器（IR）各位的定义（CAN 地址 3）

IR.7	IR.6	IR.5	IR.4	IR.3	IR.2	IR.1	IR.0
保留未用			WUI	DOI	EI	TI	RI

1）WUI：唤醒中断，当 SJA1000 离开睡眠状态时，该位将被置位。当 MCU 试图在 SJA1000 有未决中断或仍有总线活动使其进入睡眠状态时，也会产生唤醒中断。

2）DOI：数据超载中断，DOS 位的“0-1”变化时该位即被置位，即 DOS 与 DOI 的置位是同步的。

3）EI：错误中断，任何错误状态标志位或总线状态标志位的变化都会使该位置位。

4）TI：发送中断，与 TBS 同步被置位。

5）RI：接收中断，当接收缓冲区非空时该位被置位。

5．接收码寄存器 ACR

接收码寄存器决定接收滤波。当接收码（AC7～AC0）与报文高 8 位（ID10～ID3）相等时，报文通过接收滤波。如果 1 条报文通过了接收滤波，而且接收缓冲区有可用空间，那么，对应的描述符和数据场就依次进入 RXFIFO。报文正确接收之后，接收状态位置 1，如果接收中断允许（RIE＝1），那么接收中断位置 1。

6．屏蔽码寄存器 AMR

屏蔽码寄存器影响接收滤波。在屏蔽码（AMC7～AMC0）为 0 的相应位置上，接收码（AC7～AC0）与报文标志符的高 8 位（ID10～ID3）相等时，报文才可通过接收滤波。

如[（AC7～AC0）＝（ID10～ID3）]或（AMC7～AMC0）＝[11111111B]，即（ID10～ID3）与（AC.7～AC.0）按位同或后再与（AM.7～AM.0）按位或，若得 11111111B 则接收。

3.3.3　基本模式下的数据段寄存器

数据段寄存器由发送缓冲区和接收缓冲区组成。它们分别由标识符和数据域组成，发送缓冲区在运行状态下可读写。其组成结构和各位的定义见表 3-7。

表 3-7　发送缓冲区的组成结构和各位的定义

地址	名称	位名称							
10	标识符	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
11		ID2	ID.1	ID.0	RTR	DLC3	DLC2	DLC1	DL0
12～19	数据域	发送数据字节 1～8							

1．发送缓冲区

（1）标识符

标识符由 11bit 组成（ID.10~ID.0），ID.10 为最高位，在仲裁过程中，它首先被送至总线。标识符作为报文的名称，在接收器的滤波和仲裁过程中确定总线访问优先权时都要用到，标识符的二进制数值越低，其优先级越高，这是由于仲裁期间有大量的前导显性位所致。

（2）远程发送请求位

当该位置位时，SJA1000 将发送远程帧，该帧不包括数据字节，但应给出标识符所指定的数据帧正确的数据长度码。当该位复位时，SJA1000 将发送数据帧。

（3）数据长度码（DLC）

报文数据场中，字节的数目由数据场长度码编码。在远程帧开始发送时，由于 RTR 位为高，数据长度码不被考虑，这迫使发送/接收数据字节数为 0。若有两个 CAN 控制器同时开始发送相同标识符的远程帧时，数据长度码必须被正确制定，以避免总线出错（由于仲裁）。

数据字节数=8×DLC.3+4×DLC.2+2×DLC.1+ DLC.0

出于兼容性的考虑，不允许使用 0~8 以外的数据字节数。

（4）数据域

发送数据字节数由数据长度码决定，地址单元为 12 的数据字节的最高位首先被发送。

2．接收缓冲区和接收滤波

接收缓冲区的结构与发送缓冲区的结构相同，只是起始地址单元为 20。在 SJA1000 中，接收缓冲区被设计为 64B 的先入先出队列（FIFO），同一时刻 RXFIFO 中可存储报文的数目取决于报文的长度，如果没有足够的空间存储新接收的报文，将产生一个超载中断，此时，RXFIFO 中不完整的报文将被删除。

只有通过接收滤波的报文才会被写入 RXFIFO，接收滤波由 ACR 和 AMR 共同决定。

3.4　SJA1000 扩展模式下的寄存器

3.4.1　扩展模式下的寄存器

1．扩展模式下的寄存器的分类

SJA1000 上电复位后自动进入基本模式，可通过对 SJA1000 编程，使它进入扩展模式。

在扩展模式下，SJA1000 对内部寄存器资源进行了重新配置，从基本模式下的 32 个寄存器扩展到了 112 个。下面简要归纳介绍扩展模式下的 SJA1000 寄存器的结构。

扩展模式下，CAN 控制器的内部寄存器配置大致可以分为四类：第一类是 SJA1000 在基本模式和扩展模式下共有的，包括总线定时寄存器 0（BTR0）和总线定时寄存器 1（BTR1），输出控制寄存器（OCR）和时钟分频寄存器（CDR），它们完成 CAN 总线节点的基础参数设置；第二类是与错误处理密切相关的，包括仲裁丢失捕获寄存器（ALC）、错误捕获寄存器（ECC）、错误警告极限寄存器（EWLC）、接收错误计数寄存器（RXERR）、发送错误计数寄存器（TXERR）；第三类是与报文收发密切相关的，包括命令寄存器（CMR）、状态寄存器（SR）、接收寄存器（RXB）、发送寄存器（TXB）、中断寄存器（IR）、中断允许寄存器（IER）；第四类，从地址 32 起始的共有 80B 的 SJA1000 内部 RAM 及与之相关的接收报文计数器（RMC）、接收缓存起始地址寄存器（RBSA）。

特别值得用户注意的是，CAN 控制器可以分别工作在运行状态和复位状态，有些寄存器在两种状态下的特性是不同的。比如错误警告极限寄存器，在运行状态下只读而在复位状态下可读可写。另外，CAN 控制器从 16～28 这段地址空间是复用的。在运行状态下，对这段地址的读操作是对接收缓冲器进行的，写操作是对发送缓冲器进行的；在复位状态下，对这段地址的读写都是对接收码寄存器和屏蔽码寄存器进行的。

（1）第一类寄存器

这类寄存器在第 3.5 节介绍。

（2）第二类寄存器

下面介绍扩展模式下，与错误处理密切相关的几个寄存器。

1）仲裁丢失捕获寄存器：每个报文都有一个标识符，CAN 总线基于标识符来对同时发送的两个报文进行总线仲裁，仲裁失败后的那个发送节点将停止发送，仲裁丢失捕获寄存器会保存丢失仲裁的位置信息。该寄存器的内容将一直被保存下去，直至用户软件读出它的内容，这时捕获机制再次激活。如果丢失仲裁中断允许，则 CPU 将接收到一个丢失仲裁中断。

根据仲裁法则，数据帧的优先权大于远程帧，标准帧的优先权大于扩展帧。

2）错误捕获寄存器：该寄存器对 CPU 来说是只读存储器，用来保存节点所检测到的错误类型及位置信息。该寄存器的特性同仲裁捕获寄存器相似。只有该寄存器的内容被读取一次后，捕获机制才会被再次激活，捕获下一个错误信息。

3）发送错误计数寄存器：该寄存器反映了发送错误计数器的当前值。在运行状态，该寄存器对 CPU 来说只可读，在复位状态，该寄存器才可读可写。硬件复位后，该寄存器被初始化为逻辑 0。如果节点脱离总线，该寄存器将被初始化为 127。复位位被清除后，CAN 控制器将等待 128 次总线释放信号（每个信号由总线上 11bit 连续的隐性位构成）。此时，该寄存器以倒计时形式对协议规定的最小恢复时间计数。通过读取该寄存器，可以得到关于节点脱离总线后恢复的状况信息。

在脱离总线期间，对该寄存器进行从 0 到 254 的写操作，将消除脱离总线标志。复位位被清除后，CAN 控制器只需等待一个总线释放周期。若对该寄存器写入 255，将初始化 CPU 驱动的总线脱离。注意此时仍处于总线脱离期间，故写入的内容不被解释执行，当复位

请求位被清除后，发送错误计数器的内容将引起 CAN 控制器再次进入脱离总线状态。这意味着，复位请求位被再次置位，发送错误计数寄存器被初始化为 127，所有相关的状态位被置位。

（3）第三类寄存器

1）命令寄存器：相对于基本模式，该寄存器增加了一位自收请求位。在基本模式下，只能对命令位进行单个操作，而扩展模式下可对几位命令位进行组合操作。

若将 CMR.0 和 CMR.1 同时置位，则只发送一遍报文，即使出现错误或仲裁丢失也不重发。置位 CMR.0、CMR.1 和 CMR.4 也将起到同样的作用。

若将 CMR.4 和 CMR.1 同时置位，则以自发自收的形式将报文发送一遍，同样，即使出现错误或仲裁丢失也不重发。

若将 CMR.0 和 CMR.4 同时置位，则忽略 CMR.4，仍以正常方式发送报文。出现错误或仲裁丢失后，重发报文。

状态寄存器里的发送状态位一旦置位，上述各发送请求位将被自动清除。

2）接收码寄存器，接收屏蔽寄存器：接收码寄存器（ACR）和接收屏蔽寄存器（AMR）组成接收滤波器，只有通过接收滤波器的报文才能进入 FIFO 队列。扩展模式下有两种滤波方式，通过模式寄存器中 MOD.3（AFM）进行选择：单滤波方式（AFM=1）和双滤波方式（ARM=0）。

单滤波方式：该方式下，定义了一个 4B 的滤波器，滤波器与接收报文之间的位对应关系取决于当前接收报文的帧格式是标准帧还是扩展帧。只有按照正确的对应关系，每位比较通过的报文才予以接收。值得注意的是，无论是标准帧还是扩展帧，滤波器都有一些位没用上，为了与未来的产品兼容，这些位对应的接收屏蔽位都应设为“不关心”位。

双滤波方式：该方式下，定义了两组滤波器，同样，滤波器与接收报文之间的位对应关系取决于当前接收报文的帧格式是标准帧还是扩展帧。不同的是，只要报文通过一组滤波器，便会被接收。

（4）第四类寄存器

在 80B 的内部 RAM 中，有 64B 分配给了 FIFO 队列，剩余的 16B 留给了发送缓存。FIFO 队列用来存取经过滤波后的报文。接收缓存就像是 FIFO 队列的一个窗口，CPU 通过它把报文取走。接收缓存起始地址寄存器用来说明接收缓存在 FIFO 队列中的位置。取走报文后，CPU 还需发出释放接收缓存的命令。只有这样，接收缓存才能转移到 FIFO 中的下一个报文，SJA1000 才会发出接收中断，如果接收中断允许的话。在基本模式下，CPU 只能以访问接收缓存的形式接触到被接收的报文；而在扩展模式下，CPU 还能以直接访问 FIFO 队列的形式接触到被接收的报文。这正是扩展模式的灵活之所在。接收报文计数寄存器用来对已接收却未被 CPU 读取的报文进行计数。

2．扩展模式下的地址列表

CAN 控制寄存器的内部寄存器对 CPU 来说是以外部寄存器形式存在而作片内内存使用。因为 CAN 控制器可以工作于不同模式，所以必须区分不同的内部地址定义。从 CAN 地址 32 起所有的内部 RAM（80B）被映象为 CPU 的接口。PeliCAN 地址分配见表 3-8。

表 3-8　PeliCAN 地址分配

<table>
<tr><th rowspan="2">CAN
地址</th><th colspan="4">工作模式</th><th colspan="2">复位模式</th></tr>
<tr><th colspan="2">读</th><th colspan="2">写</th><th>读</th><th>写</th></tr>
<tr><td>0</td><td colspan="2">模式</td><td colspan="2">模式</td><td>模式</td><td>模式</td></tr>
<tr><td>1</td><td colspan="2">（00H）</td><td colspan="2">命令</td><td>（00H）</td><td>命令</td></tr>
<tr><td>2</td><td colspan="2">状态</td><td colspan="2">—</td><td>状态</td><td>—</td></tr>
<tr><td>3</td><td colspan="2">中断</td><td colspan="2">—</td><td>中断</td><td>—</td></tr>
<tr><td>4</td><td colspan="2">中断使能</td><td colspan="2">中断使能</td><td>中断使能</td><td>中断使能</td></tr>
<tr><td>5</td><td colspan="2">保留（00H）</td><td colspan="2">—</td><td>保留（00H）</td><td>—</td></tr>
<tr><td>6</td><td colspan="2">总线定时 0</td><td colspan="2">—</td><td>总线定时 0</td><td>总线定时 0</td></tr>
<tr><td>7</td><td colspan="2">总线定时 1</td><td colspan="2">—</td><td>总线定时 1</td><td>总线定时 1</td></tr>
<tr><td>8</td><td colspan="2">输出控制</td><td colspan="2">—</td><td>输出控制</td><td>输出控制</td></tr>
<tr><td>9</td><td colspan="2">检测</td><td colspan="2">检测[1]</td><td>检测</td><td>检测[1]</td></tr>
<tr><td>10</td><td colspan="2">保留（00H）</td><td colspan="2">—</td><td>保留（00H）</td><td>—</td></tr>
<tr><td>11</td><td colspan="2">仲裁丢失捕捉</td><td colspan="2">—</td><td>仲裁丢失捕捉</td><td>—</td></tr>
<tr><td>12</td><td colspan="2">错误代码捕捉</td><td colspan="2">—</td><td>错误代码捕捉</td><td>—</td></tr>
<tr><td>13</td><td colspan="2">错误报警限制</td><td colspan="2">—</td><td>错误报警限制</td><td>错误报警限制</td></tr>
<tr><td>14</td><td colspan="2">RX 错误计数器</td><td colspan="2">—</td><td>RX 错误计数器</td><td>RX 错误计数器</td></tr>
<tr><td>15</td><td colspan="2">TX 错误计数器</td><td colspan="2">—</td><td>TX 错误计数器</td><td>TX 错误计数器</td></tr>
<tr><td>16</td><td>RX 帧信息 SFF[2]</td><td>RX 帧信息 EFF[3]</td><td>验收代码 0</td><td>验收代码 0</td><td>验收代码 0</td><td>验收代码 0</td></tr>
<tr><td>17</td><td>RX 识别码 1</td><td>RX 识别码 1</td><td>验收代码 1</td><td>验收代码 1</td><td>验收代码 1</td><td>验收代码 1</td></tr>
<tr><td>18</td><td>RX 识别码 2</td><td>RX 识别码 2</td><td>验收代码 2</td><td>验收代码 2</td><td>验收代码 2</td><td>验收代码 2</td></tr>
<tr><td>19</td><td>RX 数据 1</td><td>RX 识别码 3</td><td>验收代码 3</td><td>验收代码 3</td><td>验收代码 3</td><td>验收代码 3</td></tr>
<tr><td>20</td><td>RX 数据 2</td><td>RX 识别码 4</td><td>验收屏蔽 0</td><td>验收屏蔽 0</td><td>验收屏蔽 0</td><td>验收屏蔽 0</td></tr>
<tr><td>21</td><td>RX 数据 3</td><td>TX 数据 1</td><td>验收屏蔽 1</td><td>验收屏蔽 1</td><td>验收屏蔽 1</td><td>验收屏蔽 1</td></tr>
<tr><td>22</td><td>RX 数据 4</td><td>TX 数据 2</td><td>验收屏蔽 2</td><td>验收屏蔽 2</td><td>验收屏蔽 2</td><td>验收屏蔽 2</td></tr>
<tr><td>23</td><td>RX 数据 5</td><td>TX 数据 3</td><td>验收屏蔽 3</td><td>验收屏蔽 3</td><td>验收屏蔽 3</td><td>验收屏蔽 3</td></tr>
<tr><td>24</td><td>RX 数据 6</td><td>TX 数据 4</td><td>保留（00H）</td><td>保留（00H）</td><td>保留（00H）</td><td>—</td></tr>
<tr><td>25</td><td>RX 数据 7</td><td>TX 数据 5</td><td>保留（00H）</td><td>保留（00H）</td><td>保留（00H）</td><td>—</td></tr>
<tr><td>26</td><td>RX 数据 8</td><td>TX 数据 6</td><td>保留（00H）</td><td>保留（00H）</td><td>保留（00H）</td><td>—</td></tr>
<tr><td>27</td><td>（FIFO RAM）[4]</td><td>TX 数据 7</td><td>保留（00H）</td><td>保留（00H）</td><td>保留（00H）</td><td>—</td></tr>
<tr><td>28</td><td>（FIFO RAM）[4]</td><td>TX 数据 8</td><td>保留（00H）</td><td>保留（00H）</td><td>保留（00H）</td><td>—</td></tr>
<tr><td>29</td><td colspan="2">RX 信息计数器</td><td colspan="2">—</td><td>RX 信息计数器</td><td>—</td></tr>
<tr><td>30</td><td colspan="2">RX 缓冲器起始地址</td><td colspan="2">—</td><td>RX 缓冲器起始地址</td><td>RX 缓冲器起始地址</td></tr>
<tr><td>31</td><td colspan="2">时钟分频器</td><td colspan="2">时钟分频器[5]</td><td>时钟分频器</td><td>时钟分频器</td></tr>
<tr><td>32</td><td colspan="2">内部 RAM 地址 0（FIFO）</td><td colspan="2">—</td><td>内部 RAM 地址 0</td><td>内部 RAM 地址 0</td></tr>
<tr><td>33</td><td colspan="2">内部 RAM 地址 1（FIFO）</td><td colspan="2">—</td><td>内部 RAM 地址 1</td><td>内部 RAM 地址 1</td></tr>
<tr><td>↓</td><td colspan="2">↓</td><td colspan="2">↓</td><td>↓</td><td>↓</td></tr>
<tr><td>95</td><td colspan="2">内部 RAM 地址 63（FIFO）</td><td colspan="2">—</td><td>内部 RAM 地址 63</td><td>内部 RAM 地址 63</td></tr>
</table>

（续）

CAN 地址	工作模式		复位模式	
	读	写	读	写
96	内部 RAM 地址 64（TX 缓冲器）	—	内部 RAM 地址 64	内部 RAM 地址 64
↓	↓	—	↓	↓
108	内部 RAM 地址 76（TX 缓冲器）	—	内部 RAM 地址 76	内部 RAM 地址 76
109	内部 RAM 地址 77（空闲）	—	内部 RAM 地址 77	内部 RAM 地址 77
110	内部 RAM 地址 78（空闲）	—	内部 RAM 地址 78	内部 RAM 地址 78
111	内部 RAM 地址 79（空闲）	—	内部 RAM 地址 79	内部 RAM 地址 79
112	（00H）	—	（00H）	—
↓	↓	↓	↓	↓
127	（00H）	—	（00H）	—

注：必须说明的是 CAN 的高端地址区的寄存器是重复的（CPU 地址的高 8bit 是不参与解码的，CAN 地址 128 和地址 0 是连续的）。

① 测试寄存器只用于产品测试。正常工作时使用该寄存器会使设备产生不可预料的行为。

② SFF=标准帧格式。

③ EFF=扩展帧格式。

④ 这些地址分配反映当前信息之后的 FIFO 的 RAM 空间。上电后的内容是随机的且包含了当前接收信息的下一条信息的开头。如果没有信息要接收，这里会出现部分旧的信息。

⑤ 一些位在复位模式中是只写的（CAN 模式、CBP、RXNTEN 的时钟关闭）。

3. 寄存器的复位值

检测到复位模式置位后，中止当前发送/接收信息而进入复位模式。当复位模式位发生‘1-0’跳变时，CAN 控制器回到模式寄存器所定义的模式。复位模式配置见表 3-9。

表 3-9　复位模式配置

寄存器	位	符号	名称	值	
				硬件复位	软件设置 MOD.0 或总线关闭时复位
模式	MOD.7～MOD.5	—	保留	0（保留）	0（保留）
	MOD.4	SM	睡眠	0（唤醒）	0（唤醒）
	MOD.3	AFM	验收滤波器	0（双向）	×
	MOD.2	STM	自检测模式	0（正常）	×
	MOD.1	LOM	只听模式	0（正常）	×
	MOD.0	RM	复位模式	0（当前）	1（当前）
命令	CMR.7～CMR.5		保留	0（保留）	0（保留）
	CMR.4	SRR	自接收模式	0（空缺）	0（空缺）
	CMR.3	CDO	清除数据超载	0（无动作）	0（无动作）
	CMR.2	RRB	释放接收缓冲器	0（无动作）	0（无动作）
	CMR.1	AT	中止发送	0（空缺）	0（空缺）
	CMR.0	AR	发送请求	0（空缺）	0（空缺）

（续）

寄存器	位	符号	名称	值	
				硬件复位	软件设置 MOD.0 或总线关闭时复位
状态	SR.7	BS	总线状态	0（总线开启）	×
	SR.6	ES	出错状态	0（ok）	×
	SR.5	TS	发送状态	1（等待空闲）	1（等待空闲）
	SR.4	RS	接收状态	1（等待空闲）	1（等待空闲）
	SR.3	TCS	发送完毕状态	1（完毕）	×
	SR.2	TBS	发送缓冲器状态	1（释放）	1（释放）
	SR.1	DOS	数据超载状态	0（空缺）	0（空缺）
	SR.0	RBS	接收缓冲器状态	0（空）	0（空）
中断	IR.7	BEI	总线出错状态	0（复位）	0（复位）
	IR.6	ALI	仲裁行动中断	0（复位）	0（复位）
	IR.5	EPI	错误消极中断	0（复位）	0（复位）
	IR.4	WUI	唤醒中断	0（复位）	0（复位）
	IR.3	DOI	数据超载中断	0（复位）	0（复位）
	IR.2	EI	错误警报中断	0（复位）	×①
	IR.1	TI	发送中断	0（复位）	0（复位）
	IR.0	RI	接收中断	0（复位）	0（复位）
中断使能	IER.7	BEIE	总线错误中断使能	×	×
	IER.6	ALIE	仲裁丢失中断使能	×	×
	IER.5	EPIE	错误消极中断使能	×	×
	IER.4	WUIE	唤醒中断使能	×	×
	IER.3	DOIE	数据超载中断使能	×	×
	IER.2	EIE	错误报警中断使能	×	×
	IER.1	TIE	发送中断使能	×	×
	IER.0	RIE	接收中断使能	×	×
总线定时 0	BTR0.7	SJW.1	同步跳转宽度 1	×	×
	BTR0.6	SJW.0	同步跳转宽度 0	×	×
	BTR0.5	BRT.5	波特率预设值 5	×	×
	BTR0.4	BRP.4	波特率预设值 4	×	×
	BTR0.3	BRP.3	波特率预设值 3	×	×
	BTR0.2	BRP.2	波特率预设值 2	×	×
	BTR0.1	BRP.1	波特率预设值 1	×	×
	BTR0.0	BRP.0	波特率预设值 0	×	×
总线定时 1	BTR1.7	SAM	采样	×	×
	BTR1.6	TSEG2.2	时间段 2.2	×	×
	BTR1.5	TSEG2.1	时间段 2.1	×	×
	BTR1.4	TSEG2.0	时间段 2.0	×	×
	BTR1.3	TSEG1.3	时间段 1.3	×	×
	BTR1.2	TSEG1.2	时间段 1.2	×	×
	BTR1.1	TSEG1.1	时间段 1.1	×	×
	BTR1.0	TSEG1.0	时间段 1.0	×	×

（续）

寄存器	位	符号	名称	值	
				硬件复位	软件设置 MOD.0 或总线关闭时复位
输出控制	OCR.7	OCTP1	输出控制晶体管 P1	×	×
	OCR.6	OCTN1	输出控制晶体管 N1	×	×
	OCR.5	OCPOL1	输出控制极性 1	×	×
	OCR.4	OCTP0	输出控制晶体管 P0	×	×
	OCR.3	OCTN0	输出控制晶体管 N0	×	×
	OCR.2	OCPOL0	输出控制极性 0	×	×
	OCR.1	OCMODE1	输出控制模式 1	×	×
	OCR.0	OCMODE0	输出控制模式 0	×	×
仲裁丢失捕捉	—	ALC	仲裁丢失捕捉	0	×
错误代码捕捉	—	ECC	错误代码捕捉	0	×
错误报警限制	—	EWLR	错误报警限制寄存器	96	×
RX 错误计数器	—	RXERR	接收错误计数器	0（复位）	×②
TX 错误计数器	—	TXERR	发送错误计数器	0（复位）	×②
TX 缓冲器	—	TXB	发送缓冲器	×	×
RX 缓冲器	—	RXB	接收缓冲器	×③	×③
ACR0～ACR3	—	ACR0～ACR3	验收代码寄存器	×	×
AMR0～AMR3	—	AMT0～AMR3	验收屏蔽寄存器	×	×
RX 信息计数器	—	RMC	RX 信息计数器	0	0
RX 缓冲器起始地址	—	RBSA	RX 缓冲器起始地址	000 0000	×
时钟分频器	—	CDR	时钟分频器地址	00000000 Intel; 00000101Motorola	×

注：1. ×表示这些寄存器或位是何值是无任何影响的。
2. 括号中是功能意义的解释。
① 在相应的中断允许时，总线关闭则错误报警中断被置位。
② 若是因为总线关闭而进入复位模式，接收错误计数器被清 0，发送错误计数器被初始化到 127 以计数 CAN 定义的包括 128 个 11bit 连续隐性（弱势）位的总线关闭恢复时间。
③ RXFIFO 的内部读/写指针复位到初始化值。连续的读 RXB 口将会得到一些未定义的值（一部分是老的信息）。如果有信息被发送，就被并行写入接收缓冲器。只有这次传送是自接收请求引起的才会产生接收中断。所以，即使接收缓冲器是空的，最后一次发送的信息也可以从接收缓冲器中读出，除非它被下一条要发送或接收的信息覆盖。硬件复位时，RXFIFO 的指针指向物理 RAM 地址‘0’。通过软件设置 CR.0 或总线关闭会使 RXFIFO 的指针指向当前有效 FIFO 的起始地址，这个地址不同于第一次释放接收器命令后的 RAM 地址‘0’。

3.4.2 扩展模式下的控制寄存器

1．模式寄存器（MOD）

模式寄存器的内容是用来改变 CAN 控制器的行为。CPU 把控制寄存器作为读/写寄存器，可以设置 MOD 寄存器的位。保留位读值为逻辑 0。模式寄存器的位功能说明见表 3-10。

表 3-10 模式寄存器的位功能说明（CAN 地址 0）

MOD.7	MOD.6	MOD.5	MOD.4	MOD.3	MOD.2	MOD.1	MOD.0
保留未用			SM	AFM	STM	LOM	RM

1）SM：睡眠模式位，若置位，在没有 CAN 中断等待和总线活动时，CAN 控制器进入睡眠模式。复位时将 CAN 控制器从睡眠状态唤醒。

2）AFM：验收滤波器模式位，置位时选择单个验收滤波器（32bit 长度），复位时选择两个验收滤波器（每个 16bit 长度）。

3）STM：自检测模式位，置位时进入自检测模式。此模式可以检测所有节点，没有任何活动的节点使用自接收命令；即使没有应答，CAN 控制器也会成功发送。复位时进入正常模式；成功发送时必须应答信号。

4）LOM：只听模式位，置位时进入只听模式。这种模式中，即使成功接收信息，CAN 控制器也不向总线发应答信号；错误计数器停止在当前值，复位时进入正常模式。

5）RM：复位模式位，置位时，SJA1000 中止当前正在接收/发送的信息，进入复位模式。此位被复位时，SJA1000 收到‘1-0’的跳变后，回到正常工作模式。

2．命令寄存器（CMR）

命令位初始化是 CAN 控制器传输层的一个动作。这个寄存器是只写的，所有位的读出值都是逻辑 0。因处理的需要，两条命令之间至少有一个内部时钟周期。内部时钟周期的频率是外部振荡器的一半。命令寄存器（CMR）的位功能说明见表 3-11。

表 3-11　命令寄存器（CMR）的位功能说明（CAN 地址 1）

CMR.7	CMR.6	CMR.5	CMR.4	CMR.3	CMR.2	CMR.1	CMR.0
保留未用			GTS	CDO	RRB	AT	TR

1）GTS：睡眠命令位，若没有未决中断和总线活动，置“1”将使 SJA1000 进入睡眠状态。进入睡眠状态后，SJA1000 的 CLKOUT 端的输出将持续 15bit 时间，以使主控制器通过此信号进入待命状态。下列三种情况下，SJA1000 将被唤醒而进入运行状态：

① GTS 位被复位；

② 出现总线活动；

③ 有未决中断使 $\overline{\text{INT}}$ 端为低。

由于总线活动而被唤醒的 SJA1000 在检测 11 个连续隐性位（总线空闲标志）前将不能接收总线信息。在复位状态下，该位不可置位。

2）CDO：清除数据超载命令位，置位时有效。

3）RRB：释放接收缓冲器命令位，置位时有效。在 MCU 读取数据后，应该释放接收缓冲器以使 MCU 接收下一批数据。该位和 CDO 可同时有效。

4）AT：中止发送命令位，该命令用于 MCU 请求挂起当前数据发送，例如有更紧急的数据要发送时。但已经在进行的数据发送不会被中止。

5）TR：发送请求位，置位时有效，复位时可取消当前发送。

3．状态寄存器（SR）

状态寄存器反映 CAN 控制器的状态。状态寄存器对 CPU 来说是只读内存，见表 3-12。

表 3-12　状态寄存器的位功能说明（CAN 地址 2）

SR.7	SR.6	SR.5	SR.4	SR.3	SR.2	SR.1	SR.0
BS	ES	TS	RS	TCS	TBS	DOS	RBS

1）BS：总线状态标志，“1”表示处于总线脱离状态。当发送错误计数器超过 255 的极

限时，SJA1000 自动将 BS 置“1”并产生一个错误中断（若允许），同时控制器进入复位工作方式，直到 MCU 清除 BS。

2）ES：错误状态标志，当至少有一个错误计数器达到或超过警告极限时该位被置“1”，若允许错误中断还将产生中断。

3）TS：发送状态标志，“1”表示 SJA1000 正在发送数据。

4）RS：接收状态标志，“1”表示 SJA1000 正在接收数据，若 TS 和 RS 同时为“0”，则表明总线空闲。

5）TCS：发送完成标志，“1”表示上次的发送已成功完成。

6）TBS：发送缓冲区状态标志，“1”表示发送缓冲区可写。若该位为“0”时，MCU 写发送缓冲区，则写入数据无效且被丢失。

7）DOS：数据超载标志，“1”表示由于 RXFIFO 没有足够的空间，收到的报文丢失。

8）RBS：接收缓冲器状态标志，“1”表示 RXFIFO 中至少有一个报文。当 MCU 读取报文后，应给出释放接收缓冲区的命令，该标志才会清零。若 RXFIFO 中还有未读报文，该位又将被置位。

4．中断寄存器（IR）

中断寄存器允许中断源的识别。当这个寄存器的一位或多位被置位时，CAN 中断将反映到 CPU。CPU 读此寄存器的时候，除了接收中断外的所有位都被复位。中断寄存器对 CPU 来说是只读存储器。其位功能说明见表 3-13。

表 3-13　中断寄存器（IR）的位功能说明（CAN 地址 3）

IR.7	IR.6	IR.5	IR.4	IR.3	IR.2	IR.1	IR.0
BEI	ALI	EPI	WUI	DOI	EI	TI	RI

1）BEI：总线错误中断，当 CAN 控制器检测到总线错误且中断使能寄存器中的 BEIE 被置位时此位被置位。

2）ALI：仲裁丢失中断，当 CAN 控制器丢失仲裁，变为接收器和中断使能寄存器的 ALIE 为被置位时，此位被置位。

3）EPI：错误消极中断，当 CAN 控制器到达错误消极状态（至少一个错误计数器超过协议规定的值 127）或从错误消极状态又进入错误活动状态以及中断寄存器的 EPIE 位被置位时此位被置 1。

4）WUI：唤醒中断，当 SJA1000 离开睡眠状态时，该位将被置位。当 MCU 试图在 SJA1000 有未决中断或仍有总线活动使其进入睡眠状态时，也会产生唤醒中断。

5）DOI：数据超载中断，DOS 位的“0-1”变化时该位即被置位，即 DOS 与 DOI 的置位是同步的。

6）EI：错误中断，任何错误状态标志位或总线状态标志位的变化都会使该位置位。

7）TI：发送中断，与 TBS 同步被置位。

8）RI：接收中断，当接收缓冲区非空时该位被置位。

5．中断使能寄存器（IER）

这个寄存器能使不同类型的中断源对 CPU 有效。这个寄存器对 CPU 来说是可读/写存储

器。其各位的功能说明见表 3-14。

表 3-14　中断使能寄存器（IER）的各位的功能说明（CAN 地址 4）

IER.7	IER.6	IER.5	IER.4	IER.3	IER.2	IER.1	IER.0
BEIE	ALIE	EPIE	WUIE	DOIE	EIE	TIE	RIE

1）BEIE：总线错误中断使能位，置位时使能总线错误中断。

2）ALIE：仲裁丢失中断使能位，置位时使能仲裁丢失中断。

3）EPIE：错误消极中断使能位，置位时使能错误消极中断。

4）WUIE：唤醒中断使能位，置位时使能唤醒中断。

5）DOIE：数据超载中断使能位，置位时使能数据超载中断。

6）EIE：错误中断使能位，置位时使能错误中断。

7）TIE：发送中断使能位，置位时使能发送中断。

8）RIE：接收中断使能位，置位时使能接收中断。此位对接收中断和外部中断输出 $\overline{\text{INT}}$ 有直接的影响。如果 RIE 被清 0 且没有其他中断被挂起，外部 $\overline{\text{INT}}$ 引脚电平会立即变高。

6．仲裁丢失捕捉寄存器（ALC）

这个寄存器包括了仲裁丢失的位置信息。仲裁丢失捕捉寄存器对 CPU 来说是只读存储器。读保留位的值为 0。其各位的功能说明见表 3-15。

表 3-15 仲裁丢失捕捉寄存器（ALC）的各位的功能说明（CAN 地址 11）

ALC.7	ALC.6	ALC.5	ALC.4	ALC.3	ALC.2	ALC.1	ALC .0
保留未用			BITNO4	BITNO3	BITNO2	BITNO1	BITNO0

仲裁丢失时，会产生相应的仲裁丢失中断（中断允许）。同时，位流处理器的当前位置被捕捉送入仲裁丢失捕捉寄存器。一直到用户通过软件读这个值，寄存器中的内容都不会变。随后，捕捉机制又被激活了。

读中断寄存器时，中断寄存器中相应的中断标志位被清除，直到仲裁丢失捕捉寄存器被读一次之后，新的仲裁丢失中断才有效。

仲裁丢失捕捉寄存器的 Bit4～Bit0 的功能见表 3-16。

表 3-16　仲裁丢失捕捉寄存器的 Bit4～Bit0 的功能

位					十进制值	功　　能
ALC.4	ALC.3	ALC.2	ALC.1	ALC.0		
0	0	0	0	0	0	仲裁丢失在识别码的 bit1
0	0	0	0	1	1	仲裁丢失在识别码的 bit2
0	0	0	1	0	2	仲裁丢失在识别码的 bit3
0	0	0	1	1	3	仲裁丢失在识别码的 bit4
0	0	1	0	0	4	仲裁丢失在识别码的 bit5
0	0	1	0	1	5	仲裁丢失在识别码的 bit6
0	0	1	1	0	6	仲裁丢失在识别码的 bit7

（续）

位					十进制值	功　能
ALC.4	ALC.3	ALC.2	ALC.1	ALC.0		
0	0	1	1	1	7	仲裁丢失在识别码的 bit8
0	1	0	0	0	8	仲裁丢失在识别码的 bit9
0	1	0	0	1	9	仲裁丢失在识别码的 bit10
0	1	0	1	0	10	仲裁丢失在识别码的 bit11
0	1	0	1	1	11	仲裁丢失在 SRTR 位①
0	1	1	0	0	12	仲裁丢失在 IDE 位
0	1	1	0	1	13	仲裁丢失在识别码的 bit12②
0	1	1	1	0	14	仲裁丢失在识别码的 bit13②
0	1	1	1	1	15	仲裁丢失在识别码的 bit14②
1	0	0	0	0	16	仲裁丢失在识别码的 bit15②
1	0	0	0	1	17	仲裁丢失在识别码的 bit16②
1	0	0	1	0	18	仲裁丢失在识别码的 bit17②
1	0	0	1	1	19	仲裁丢失在识别码的 bit18②
1	0	1	0	0	20	仲裁丢失在识别码的 bit19②
1	0	1	0	1	21	仲裁丢失在识别码的 bit20②
1	0	1	1	0	22	仲裁丢失在识别码的 bit21②
1	0	1	1	1	23	仲裁丢失在识别码的 bit22②
1	1	0	0	0	24	仲裁丢失在识别码的 bit23②
1	1	0	0	1	25	仲裁丢失在识别码的 bit24②
1	1	0	1	0	26	仲裁丢失在识别码的 bit25②
1	1	0	1	1	27	仲裁丢失在识别码的 bit26②
1	1	1	0	0	28	仲裁丢失在识别码的 bit27②
1	1	1	0	1	29	仲裁丢失在识别码的 bit28②
1	1	1	1	0	30	仲裁丢失在识别码的 bit29②
1	1	1	1	1	31	仲裁丢失在 RTR 位②

注：仲裁丢失的二进制编码结构位的号码。
① 标准帧信息的 RTR 位。
② 只使用于扩展帧信息。

7．错误代码捕捉寄存器（ECC）

这个寄存器包含了总线错误的类型和位置信息。错误代码捕捉寄存器对 CPU 来说是只读内存。其位功能说明见表 3-17。

表 3-17　错误代码捕捉寄存器的位功能说明（CAN 地址 12）

ECC.7	ECC.6	ECC.5	ECC.4	ECC.3	ECC.2	ECC.1	ECC.0
ERRC1	ERRC0	DIR	SEG4	SEG3	SEG2	SEG1	SEG0

ECC.7 和 ECC.6 的解释见表 3-18，ECC.4～ECC.0 的解释见表 3-19。

表 3-18　ECC.7 和 ECC.6 的功能说明

位 ECC.7	位 ECC.6	功　　能
0	0	位错
0	1	格式错
1	0	填充错
1	1	其他错误

总线发生错误时被迫产生相应的错误中断（中断允许时）。同时，位流处理器的当前位置被捕捉送入错误代码捕捉寄存器。其内容直到用户通过软件读出时都是不变的。读出后，捕捉机制又被激活了。访问中断寄存器期间，中断寄存器中的中断标志位被清除。新的总线中断直到捕捉寄存器被读出一次才可能有效。

8．错误报警限制寄存器（EMLR）

错误报警限制在这个寄存器中被定义。默认值（硬件复位时）是 96。复位模式中，此寄存器对 CPU 来说是可读/写的。工作模式中是只读的。其位说明详见表 3-20。

注意：只有之前进入复位模式，EWLR 才有可能被改变。直到复位模式被再次取消后，才有可能发生错误状态的改变（见状态寄存器）和由新的寄存器内容引起的错误报警中断。

表 3-19　ECC.4～ECC.0 的功能说明

位 ECC.4	位 ECC.3	位 ECC.2	位 ECC.1	位 ECC.0	功　　能
0	0	0	1	1	帧开始
0	0	0	1	0	ID.28～ID.21
0	0	1	1	0	ID.20～ID.18
0	0	1	0	0	SRTR 位
0	0	1	0	1	IDE 位
0	0	1	1	1	ID.17～ID.13
0	1	1	1	1	ID.12～ID.5
0	1	1	1	0	ID.4～ID.0
0	1	1	0	0	RTR 位
0	1	1	0	1	保留位 1
0	1	1	0	1	保留位 0
0	1	0	1	1	数据长度代码
0	1	0	1	0	数据区
0	1	0	0	0	CRC 序列
1	1	0	0	0	CRC 定义符
1	1	0	0	1	应答通道
1	1	0	1	1	应答定义符
1	1	0	1	0	帧结束
1	0	0	1	0	中止
1	0	0	0	1	活动错误标志
1	0	1	1	0	消极错误标志
1	0	0	1	1	支配（控制）位误差
1	0	1	1	1	错误定义符
1	1	1	0	0	超载标志

表 3-20　错误报警寄存器（EWLR）的位说明（CAN 地址 13）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
EWL.7	EWL.6	EWL.5	EWL.4	EWL.3	EWL.2	EWL.1	EWL.0

9．RX 错误计数寄存器（RXERR）

RX 错误计数寄存器反应了接收错误计数器的当前值。硬件复位后寄存器被初始化为 0。工作模式中，对 CPU 来说是只读的。只有在复位模式中才可以写访问此寄存器。

注意，只有之前进入复位模式，才有可能由 CPU 迫使 RX 错误计数器发生改变。直到复位模式被取消后，错误状态的改变（见状态寄存器）、错误报警和由新的寄存器内容引起的错误中断才可能有效。RX 错误计数寄存器各位的功能说明详见表 3-21。

表 3-21　RX 错误计数寄存器（RXERR）各位的功能说明（CAN 地址 14）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
RXERR.7	RXERR.6	RXERR.5	RXERR.4	RXERR.3	RXERR.2	RXERR.1	RXERR.0

10．TX 错误计数寄存器（TXERR）

TX 错误计数寄存器反映了发送错误计数器的当前值。其各位的功能说明详见表 3-22。

工作模式中，这个寄存器对 CPU 是只读内存。复位模式中才可以写访问这个寄存器。硬件复位后，寄存器初始化为 0。如果总线关闭，TX 错误计数器被初始化为 127 来计算总线定义的最小时间（128 个总线空闲信号）。这段时间里读 TX 错误计数器将反映出总线关闭恢复的状态信息。

如果总线关闭是激活的，写访问 TXERR 的 0～254 单元会清除总线关闭标志，复位模式被清除后控制器会等待一个 11bit 的连续隐性（弱势）位（总线空闲）。

表 3-22　TX 错误计数寄存器（TXERR）的各位的功能说明（CAN 地址 15）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
TXERR.7	TXERR.6	TXERR.5	TXERR.4	TXERR.3	TXERR.2	TXERR.1	TXERR.0

向 TXERR 写入 255 会初始化 CPU 驱动的总线关闭事件。只有之前进入复位模式，才有可能发生 CPU 引起的 TX 错误计数器内容的改变。直到复位模式被再次取消，错误或总线状态的改变（见状态寄存器）、错误报警和由新的寄存器内容引起的错误中断才有可能有效。离开复位模式后，就像总线错误引起的一样，给出新的 TX 计数内容且总线关闭被同样地执行。这意味着重新进入复位模式，TX 错误计数器被初始化到 127，RX 计数器被清 0，所有的相关状态和中断寄存器位被置位。

复位模式的清除将会执行协议规定的总线关闭恢复序列（等待 128 个总线空闲信号）。

如果在总线关闭恢复（TXERR>0）之前又进入复位模式，总线关闭保持有效且 TXERR 被锁定。

11．验收滤波器

在验收滤波器的帮助下，只有当接收信息中的识别位和验收滤波器预定义的值相等时，CAN 控制器才允许将已接收信息存入 RXFIFO。

验收滤波器由验收代码寄存器（ACRn）和验收屏蔽寄存器（AMRn）定义。要接收的信息的位模式在验收代码寄存器中定义。相应的验收屏蔽寄存器允许定义某些位为“不影响”（即可为任意值）。

有两种不同的过滤模式可在模式寄存器中选择：

单滤波器模式（AFM 位是 1）和双滤波器模式（AFM 位是 0）。

（1）单滤波器配置

这种滤波器配置可以定义一个长滤波器（4B）。滤波器字节和信息字节之间位的对应关系取决于当前接收帧格式。

1）标准帧：如果接收的是标准帧格式的信息，在验收滤波中只使用前两个数据字节来存放包括 RTR 位的完整的识别码。如果由于置位 RTR 位而导致没有数据字节，或因为设备相应的数据长度代码而没有或只有一个数据字节，信息也会被接收的。对于一个成功接收的信息，所有单个位的比较后都必须发出接收信号。

注意：AMR1 和 ACR1 的低四位是不用的。为了和将来的产品兼容，这些位可通过设置 AMR1.3、AMR1.2、AMR1.1 和 AMR1.0 为 1 而定为“不影响”。

2）扩展帧：如果接收的信息是扩展帧格式的，包括 RTR 位的全部识别码将被接收过滤使用。

为了成功接收信息，每个位的比较后都必须发出接收信号。必须注意的是，AMR3 的最低两位和 ACR3 是不用的。为了和将来的产品兼容，这些位应该通过置位 AMR3.1 和 AMR3.0 来定为“不影响”。

（2）双滤波器的配置

这种配置可以定义两个短滤波器。一条接收的信息要和两个滤波器比较来决定是否放入接收缓冲器中。至少有一个滤波器发出接收信号，接收的信息才有效。滤波器字节和信息字节之间位的对应关系取决于当前接收的帧格式。

1）标准帧：如果接收的是标准帧的信息，被定义的两个滤波器是不一样的。第一个滤波器比较包括 RTR 位的整个标准识别码和信息的第一个数据字节。第二个滤波器只比较包括 RTR 位的整个标准识别码。

为了成功接收信息，所有单个位的比较时应至少有一个滤波器表示接收。RTR 位置位或数据长度代码是 0 时表示没有数据字节存在。无论怎样，只要从开始到 RTR 位的部分都被表示接收，信息就可以通过滤波器 1。

如果没有向滤波器请求数据字节过滤，AMR1 和 AMR3 的低四位必须被置为 1（不影响）。当使用包括 RTR 位的整个标准识别码时，两个滤波器都同样工作。

2）扩展帧：如果接收到扩展帧信息，定义的两个滤波器是相同的。两个滤波器都只比较扩展识别码的前两个字节。

为了能成功接收信息，所有单个位的比较时至少有一个滤波器表示接收。

具体详见第 7.2 节。

12．RX 信息计数器（RMC）

RMC 寄存器（CAN 地址 29）反映了 RXFIFO 中可用的信息数目。其值每次接收时加 1，每次释放接收缓冲器时减 1。每次复位后，该寄存器清 0。其各位的功能说明

见表 3-23。

表 3-23　RX 信息计数器（RMC）各位的功能说明（CAN 地址 29）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
0①	0①	0①	RMC.4	RMC.3	RMC.2	RMC.1	RMC.0

① 此位不能被写。读这个寄存器时结果总是 0。

13．RX 缓冲器起始地址寄存器（RBSA）

RBSA 寄存器（CAN 地址 30）反映了当前可用来存储位于接收缓冲器窗口中的信息的内部 RAM 地址。这条信息可以帮助说明内部 RAM 的内容。起始于 CAN 地址 32 的内部 RAM 地址区可以被 CPU 读/写访问（复位模式只能写）。

例子：如果 RBSA 被设置为 24（十进制），当前在接收缓冲器窗口中的可视信息被存储在内部 RAM，起始地址 24。因为 RAM 也被直接列入 CAN 地址空间（起始地址 32，等于 RAM 地址 0），所以这条信息也可以用 CAN 地址 56 及随后字节地址访问，即 CAN 地址=RBSA+32=24+32=56。

如果信息超过 RAM 地址 63，会从地址 0 继续。

当 FIFO 中至少有一条可用信息时就会执行释放接收缓冲器命令。RBSA 在下一条信息开始的时候更新。硬件复位时，指针初始化为“00H”。软件复位（设置为复位模式）时，指针保持原值，但 FIFO 被清空；这就意味着 RAM 的内容是不变的，但下一条接收的（或传送的）信息将会覆盖当前在接收缓冲器窗口的可视信息。

RX 缓冲器起始地址寄存器在工作模式中是只读的，在复位模式中是可读/写的。必须注意写访问 RBSA 首次有效是在下一个内部时钟频率的上升沿，内部时钟频率是外部振荡器的 1/2。其各位的功能说明见表 3-24。

表 3-24　RX 缓冲器起始地址寄存器（RBSA）各位的功能说明（CAN 地址 30）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
0①	0①	RBSA.5	RBSA.4	RBSA.3	RBSA.2	RBSA.1	RBSA.0

① 此位不能写，此寄存器的读出值总是 0。

3.4.3　扩展模式下的数据段寄存器

本节主要介绍扩展模式下的数据段寄存器包括发送缓冲器、接收缓冲器。

1．发送缓冲器

发送缓冲器的全部列表如图 3-3 所示。发送缓冲器允许定义长达 8 个数据字节的发送信息，在使用中务必分清标准帧格式（SFF）和扩展帧格式（EFF）的配置。

（1）发送缓冲器列表

发送缓冲器被分为描述符区和数据区，描述符区的第一个字节是帧信息字节（帧信息）。它说明了帧格式（SFF 或 EFF）、远程或数据帧和数据长度。SFF 有两个字节的识别码，EFF 有 4B 的识别码。数据区最多长 8 个数据字节。发送缓冲器长 13B，在 CAN 地址的 16～28。

注意：使用 CAN 地址的 96～108 可以直接访问发送缓冲器的 RAM。这个 RAM 区是为发送缓冲器保留的。下面 3B 是通用的：CAN 地址的 109、110 和 111。

CAN 地址	标准帧	CAN 地址	扩展帧
16	TX 帧信息	16	TX 帧信息
17	TX 识别码 1	17	TX 识别码 1
18	TX 识别码 2	18	TX 识别码 2
19	TX 数字节 1	19	TX 识别码 3
20	TX 数字节 2	20	TX 识别码 4
21	TX 数字节 3	21	TX 数字节 1
22	TX 数字节 4	22	TX 数字节 2
23	TX 数字节 5	23	TX 数字节 3
24	TX 数字节 6	24	TX 数字节 4
25	TX 数字节 7	25	TX 数字节 5
26	TX 数字节 8	26	TX 数字节 6
27	未使用	27	TX 数字节 7
28	未使用	28	TX 数字节 8

图 3-3　标准帧和扩展帧格式在发送缓冲器里的列表

（2）发送缓冲器的描述符区

发送缓冲器位的列表见表 3-25～表 3-33。给出的配置是和接收缓冲器列表相匹配的。

表 3-25　TX 帧信息（SFF）（CAN 地址 16）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
FF①	RTR②	×③	×③	DLC.3④	DLC.2④	DLC.1④	DLC.0④

① 帧格式。
② 远程发送请求。
③ 不影响，推荐与接收缓冲器（0）一致，以防使用自接收设备（自测模式）。
④ 数据长度代码位。

表 3-26　TX 识别码 1（SFF）（CAN 地址 17）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
ID.28①	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

① ID.×表示识别码的×位。

表 3-27　TX 识别码 2（SFF）（CAN 地址 18）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
ID.20	ID.19	ID.18	×①	×②	×②	×②	×②

注：ID.×表示识别码的×位。
① 不影响，推荐与接收缓冲器（RTR）一致，以防使用自接收设备（自测模式）。
② 不影响，推荐与接收缓冲器（0）一致，以防使用自接收设备（自测模式）。

表 3-28　TX 帧信息（EFF）（CAN 地址 16）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
FF①	RTR②	×③	×③	DLC.3④	DLC.2④	DLC.1④	DLC.0④

① 帧格式。
② 远程发送请求。
③ 不影响，推荐与接收缓冲器（0）一致，以防使用自接收设备（自测模式）。
④ 数据长度代码位。

表 3-29　TX 识别码 1（EFF）（CAN 地址 17）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

表 3-30　TX 识别码 2（EFF）（CAN 地址 18）①

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

① ID.×表示识别码的×位。

表 3-31　TX 识别码 3（EFF）（CAN 地址 19）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

表 3-32　TX 识别码 4（EFF）（CAN 地址 20）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
ID.4	ID.3	ID.2	ID.1	ID.0	×①	×②	×②

① 不影响，推荐与接收缓冲器（RTR）一致，以防使用自接收设备（自测模式）。
② 不影响，推荐与接收缓冲器（0）一致，以防使用自接收设备（自测模式）。

表 3-33　帧格式（FF）和远程发送请求（RTR）位

位	值	功　　能
FF	1	（EFF）CAN 控制器将发送扩展帧格式
	0	（SFF）CAN 控制器将发送标准帧格式
RTR	1	（远程）CAN 控制器将发送远程帧
	0	（数据）CAN 控制器将发送数据帧

（3）数据长度代码（DLC）

数据区的信息字节长度由数据长度代码编制。在远程帧发送开始时由于 RTR 位被置位（远程），数据长度代码是不被考虑的。这使接收/发送的数据字节数目为 0。如果有两个 CAN 控制器使用同一个识别码同时启动远程帧传送，数据长度代码必须正确说明以避免总线错误。

数据字节长度范围是 0～8，编码形式如下：

数据字节数=8×DLC.3+4×DLC.2+2×DLC.1+DLC.0

为了兼容，大于 8 的数据长度代码是不可用的。如果大于 8，将以 8B 计。

（4）识别码（ID）

标准帧格式（SFF）的识别码有 11bit（ID.28~ID.18），扩展帧格式的识别码有 29bit

（ID.28～ID.0）。ID.28 是最高位，在总线仲裁过程中最先发送到总线上。识别码就像信息的名字一样，在验收滤波器使用中，而且在仲裁过程中决定了总线访问的优先权。识别码的二进值越低优先权越高。这是由于仲裁时有大量的前导支配位。

（5）数据区

发送的字节数取决于数据长度代码。最先发送的是在 CAN 地址 19（SFF）或 21（EFF）的数据字节 1 的最高位。

2．接收缓冲器

接收缓冲器的列表与前面一节讲述的发送缓冲器很相似。接收缓冲器是 RXFIFO 的可访问部分，位于 CAN 地址的 16～28。每条信息都分为描述符和数据区。

接收缓冲器的位列表见表 3-34～表 3-36（SFF）和表 3-37～表 3-41（EFF）。所选配置是与接收缓冲器列表相匹配的。

表 3-34　RX 帧信息（SFF）（CAN 地址 16）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
FF①	RTR②	0	0	DLC.3③	DLC.2③	DLC.1③	DLC.0③

① 帧格式。
② 远程发送请求。
③ 数据长度代码位。

表 3-35　RX 识别码 1（SFF）（CAN 地址 17）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

表 3-36　RX 识别码 2（SFF）（CAN 地址 18）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
ID.20	ID.19	ID.18	RTR①	0	0	0	0

① 远程发送请求。

表 3-37　RX 帧信息（EFF）（CAN 地址 16）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
FF①	RTR②	0	0	DLC.3③	DLC.2③	DLC.1③	DLC.0③

① 帧格式。
② 远程发送请求。
③ 数据长度代码位。

表 3-38　RX 识别码 1（EFF）（CAN 地址 17）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

表 3-39　RX 识别码 2（EFF）（CAN 地址 18）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

表 3-40　RX 识别码 3（EFF）（CAN 地址 19）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

表 3-41　RX 识别码 4（EFF）（CAN 地址 20）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
ID.4	ID.3	ID.2	ID.1	ID.0	RTR①	0	0

① 远程发送请求。

在帧信息字节中的接收字节长度代码代表实际发送的数据长度代码，它有可能大于 8（取决于发送器）。无论如何，最大接收数据字节数是 8。在读接收缓冲器中的信息时应当考虑。

如图 3-4，RXFIFO 共有 64 个信息字节的空间。一次可以存储多少条信息取决于数据的长度。如果 RXFIFO 中没有足够的空间来存储新的信息，CAN 控制器会产生数据超载帧，此时信息有效且接收检测为肯定。发生数据超载情况时，已部分写入 RXFIFO 的信息将被删除。这种情况可以通过状态寄存器和数据超限中断（中断允许）反映到 CPU。

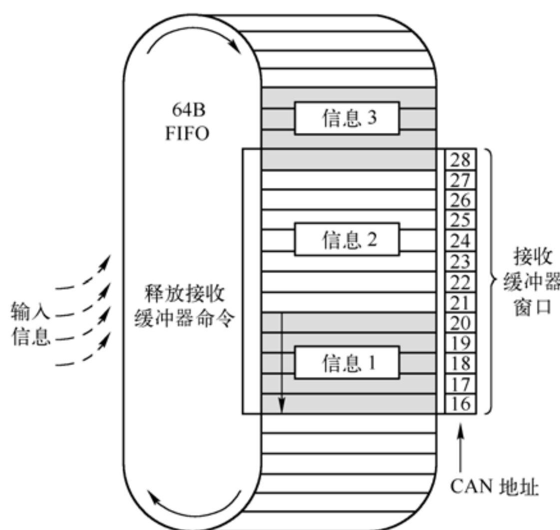


图 3-4　RXFIFO 中的信息格式存储举例

3.5　两种模式的公共寄存器

1．总线定时寄存器 0（BTR0）

额定（标称）的位定时由 3 个互不重叠的同步段（SYNC-SEG）、相位缓冲段 1（PHASE-SEG1）、相位缓冲段 2（PHSAE-SEG2）组成，这 3 个时间段是 $t_{SYNC\text{-}SEG}$、t_{TSEG1} 和 t_{TSEG2}（如图 3-5 示）由总线定时寄存器 0 和总线定时寄存器 1 定义。算术上额定位周期 t_{Bit} 是 3 个时间段的和：

$$t_{Bit} = t_{SYNC\text{-}SEG} + t_{TSEG1} + t_{TSEG2}$$

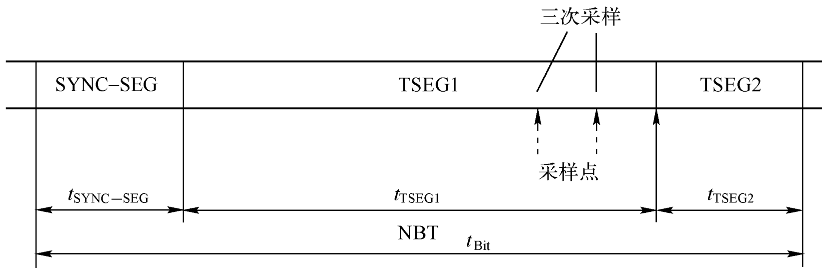


图 3-5　CAN 位定时段

总线定时寄存器 0 定义了波特率预设值（BRP）和同步跳转宽度（SJW）的值，见表 3-42。复位模式有效时这个寄存器是可以被访问（读/写）的。如果选择的是 PeliCAN 模式，此寄存器在工作模式中是只读的。在 BasicCAN 模式中总是“FFH”。

表 3-42　总线定时寄存器 0（BTR0）的位功能说明（CAN 地址 6）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
SJW.1	SJW.0	BRP.5	BRP.4	BRP.3	BRP.2	BRP.1	BRP.0

（1）波特率预设值（BRP）

CAN 系统时钟 t_{SCL} 的周期是可编程的，而且决定了相应的位时序。CAN 系统时钟由如下公式计算：

$t_{SCL}=2\times t_{CLK}\times$（32×BRP.5+16×BRP.4+8×BRP.3+4×BRP.2+2×BRP.1+BRP.0+1）

式中，t_{CLK}=XTAL 的时钟周期=$1/f_{XTAL}$。

（2）同步跳转宽度（SJW）

为了补偿在不同总线控制器的时钟振荡器之间的相位偏移，任何总线控制器必须在当前传送的相关信号边沿重新同步。同步跳转宽度定义了每一位周期可以被重新同步缩短或延长的时钟周期的最大数目，同步跳转宽度计算公式如下：

$t_{SJW}=t_{SCL}\times$（2×SJW.1+SJW.0+1）

SJW 段并不是位周期的一段，只是定义重同步事件中被增长或缩短的位周期的最大 TQ 数量。

SJW 的范围为 1～4t_{SCL}。

2．总线定时寄存器 1（BTR1）

总线定时寄存器 1 定义了每个位周期的长度、采样点的位置和每个采样点的采样数目，见表 3-4。在复位模式中，这个寄存器可以被读/写访问。在 PeliCAN 模式的工作模式中，这个寄存器是只读的。在 BasicCAN 模式中总是“FFH”。

表 3-43　总线定时寄存器 1（BTR1）的各位功能说明（CAN 地址 7）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
SAM	TSEG2.2	TSEG2.1	TSEG2.0	TSEG1.3	TSEG1.2	TSEG1.1	TSEG1.0

（1）采样（SAM）

CAN 协议允许用户指定位采样模式（SAM），分别是单次采样和三次采样模式（在 3 个采样结果中选出 1 个）。在单次采样模式中，采样点是在 TSEG1 段的末端。而三次采样模式比单次采样多取两个采样点，它们在 TSEG1 段的末端的前面，相邻之间相差一个 TQ。

表 3-44　采样（SAM）寄存器的各位功能说明

位	值	功　能
SAM	1	三倍，总线采样三次，建议在低/中速总线（A 和 B 级）上使用，这对过滤总线上的毛刺波是有益的。
	0	单倍，总线采样一次，建议使用在高速总线上（SAEC 级）。

（2）时间段 1（TSEG1）和时间段 2（TSEG2）

（TSEG1）和（TSEG2）决定了每一位的时钟数目和采样点的位置，这里：

$t_{SYNC\text{-}SEG}=1\times t_{SCL}$

$t_{TSEG1}=t_{SCL}\times$（8×TSEG1.3+4×TSEG1.2+2×TSEG1.1+TSEG1.0+1）

$t_{TSEG2}=t_{SCL}\times$（4×TSEG2.2+2×TSEG2.1+TSEG2.0+1）

每个时间段都用整个基本时间单位来表示，这个时间单位就叫时间份额（TQ）。时间份额的持续时间是 CAN 系统时钟的一个周期 t_{SCL}，是从振荡器时钟周期（t_{CLK}）取得，如图 3-6 所示。

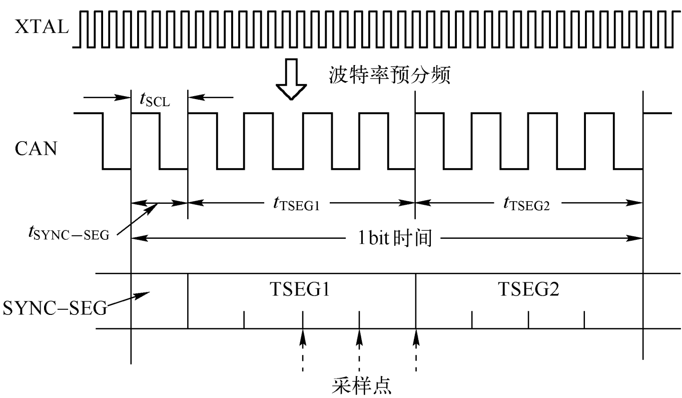


图 3-6　CAN 控制器的位周期

注意：TSEG2 必须选择 ≥2（单次采样模式）和≥3（3 次采样模式）。

3．输出控制寄存器（OCR）

输出控制寄存器实现了由软件控制不同输出驱动配置的建立，各位功能说明见表 3-45。在复位模式中此寄存器可被读/写访问。在 PeliCAN 模式中，这个寄存器是只读的。在 BasicCAN 模式中总是“FFH”。

表 3-45　输出控制寄存器（OCR）的各位功能说明（CAN 地址 8）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
OCTP1	OCTN1	OCPOL1	OCTP0	OCTN0	OCPOL0	OCMODE1	OCMODE0

当 SJA1000 在睡眠模式中时，TX0 和 TX1 引脚根据输出控制寄存器的内容输出隐性的

电平。在复位状态（复位请求=1）或外部复位引脚/$\overline{\text{RST}}$被拉低时，输出 TX0 和 TX1 悬空。

发送的输出阶段可以有不同的模式。表 3-46 列出了输出控制寄存器的设置。

表 3-46　OCMODE 位的说明

OCMODE1	OCMODE0	说　明
0	0	双相输出模式
0	1	测试输出模式
1	0	正常输出模式
1	1	时钟输出模式

（1）正常输出模式

正常模式中位序列（TXD）通过 TX0 和 TX1 送出。输出驱动引脚 TX0 和 TX1 的电平取决于被 OCTPX，OCTNX（悬空，上拉，下拉，推挽）编程的驱动器的特性和被 OCPOLX 编程的输出端极性。

（2）时钟输出模式

TX0 引脚在该模式中和正常模式是相同的。但是，TX1 上的数据流被发送时钟（TXCLK）代替了。发送时钟（不翻转）的上升沿标志着一位的开始。时钟脉冲宽度是 $1\times t_{SCL}$。

（3）双相输出模式

相对于正常输出模式，这里的位代表着时间的变化和触发。如果总线控制器被发送端从总线上通电退耦，则位流不允许含有直流元件。这一点的总结见表 3-47。在隐性位无效（悬空）期间，支配位轮流使用 TX0 或 TX1 电平发送。例如，第一位在 TX0 上发送，第二位在 TX1 上发送，第三位在 TX0 上发送等等，依此类推。

（4）测试输出模式

在测试输出模式中，RX 上的电平在下一个系统时钟的上升沿映射到 TXn 上，系统时钟（$f_{OSC}/2$）与输出控制寄存器中定义的极性一致。表 3-47 显示了输出控制寄存器的位和输出脚 TX0 和 TX1 的关系。位序列（TXD）通过 TX0 和 TX1 发送。输出驱动引脚上的电平取决于被 OCTPX，OCTNX（悬空，上拉，下拉，推挽）编程的驱动器的特性和被 OCPOLX 编程的输出端极性。

4．时钟分频寄存器（CDR）

时钟分频寄存器为微控制器控制 CLKOUT 的频率以及屏蔽 CLKOUT 引脚。而且它还控制着 TX1 上的专用接收中断脉冲、接收比较通道和 BasicCAN 模式与 PeliCAN 模式的选择。硬件复位后寄存器的默认状态是 Motorola 模式（0000 0101，12 分频）和 Intel 模式（0000 0000，2 分频）。

软件复位（复位请求/复位模式）时，此寄存器不受影响。

保留位（CDR.4）总是 0。应用软件总是向此位写 0 以与将来可能使用此位的特性兼容。

表 3-47　输出引脚配置

驱动	TXD	OCTPX	OCTNX	OCPOLX	TPX②	TNX③	TXn④
悬空	×①	0	0	×	关	关	悬空
上拉	0	0	1	0	关	开	低

（续）

驱动	TXD	OCTPX	OCTNX	OCPOLX	TPX[②]	TNX[③]	TXn[④]
上拉	1	0	1	0	关	关	悬空
	0	0	1	1	关	关	悬空
	1	0	1	1	关	开	低
下拉	0	1	0	0	关	关	悬空
	1	1	0	0	开	关	高
	0	1	0	1	开	关	高
	1	1	0	1	关	关	悬空
上拉	0	1	1	0	关	开	低
	1	1	1	0	开	关	高
	0	1	1	1	开	关	高
	1	1	1	1	关	开	低

① ×=不影响。
② TPX 是片内输出发送器 X，连接 VDD。
③ TNX 是片内输出发送器 X，连接 VSS。
④ TXn 是在引脚下 TX0 或 TX1 上的串行输出电平。当 TXD=0 和 TXD 连续是 1 时，CAN 总线上的输出电平必须是本地的。

表 3-48　时钟分频寄存器（CDR）各位的功能说明（CAN 地址 31）

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
CAN 模式	CBP	RXINTEN	0[①]	关闭时钟	CD.2	CD.1	CD.0

① 此位不能被写。读值总为 0。

（1）CD.2～CD.0

复位模式和工作模式中一样，CD.2～CD.0 可以无限制访问。这些位用来定义外部 CLKOUT 引脚上的频率。可选频率一览表见表 3-49。

表 3-49　CLKOUT 频率选择

CD.2	CD.1	CD.0	时钟频率
0	0	0	$f_{osc}/2$
0	0	1	$f_{osc}/4$
0	1	0	$f_{osc}/6$
0	1	1	$f_{osc}/8$
1	0	0	$f_{osc}/10$
1	0	1	$f_{osc}/12$
1	1	0	$f_{osc}/14$
1	1	1	f_{osc}

注：f_{osc} 是外部振荡器（XTAL）频率。

（2）时钟关闭

设置这一位可禁止 SJA1000 的外部 CLKOUT 引脚。只有在复位模式中才可以写访问。如果置位此位，CLKOUT 引脚在睡眠模式中是低而其他情况下是高。

（3）RXINTEN

此位允许 TX1 引脚用来做专用接收中断输出。当一条已接收的信息成功地通过验收滤波器，一位时间长度的接收中断脉冲就会在 TX1 引脚输出（帧的最后一位期间）。发送输出阶段应该工作在正常输出模式。极性和输出驱动可以通过输出控制寄存器编程。复位模式中只能写访问。

（4）CBP

只能在复位模式中置位 CDR.6 来中止 CAN 输入比较器。这主要用于 SJA1000 外接发送接收电路时。此时内部延时减少，这将会导致总线最大长度增加。如果 CBP 被置位，只有 RX0 被激活。没有被使用的 RX1 输入应被连接到一个确定的电平（例如 V_{SS}）。

（5）CAN 模式

CDR.7 定义了 CAN 模式。如果 CDR.7 是 0，CAN 控制器工作于 BasicCAN 模式。否则，CAN 控制器工作于 PeliCAN 模式。只有在复位模式中是可写的。

3.6 SJA1000 的读写时序分析

CAN 总线控制器的读/写时序和 MCS-51 单片机类似，数据总线与地址总线复用，具体如图 3-7 和图 3-8 所示，与读写有关的信号说明见表 3-50。

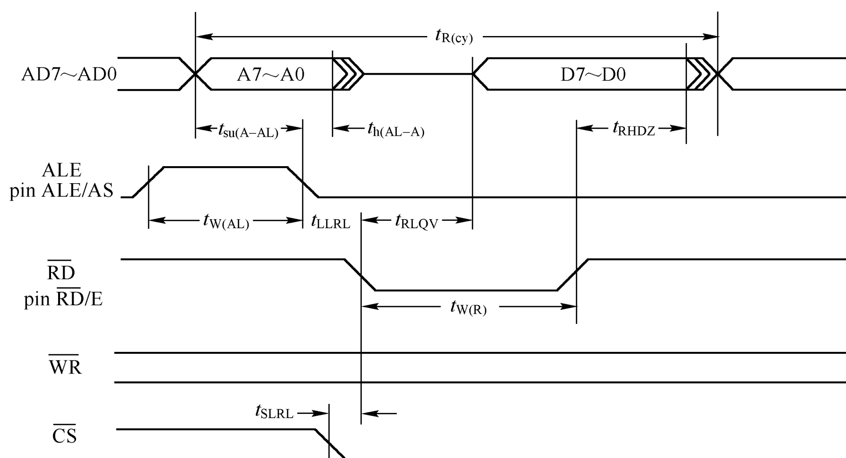


图 3-7　SJA1000 读周期时序图

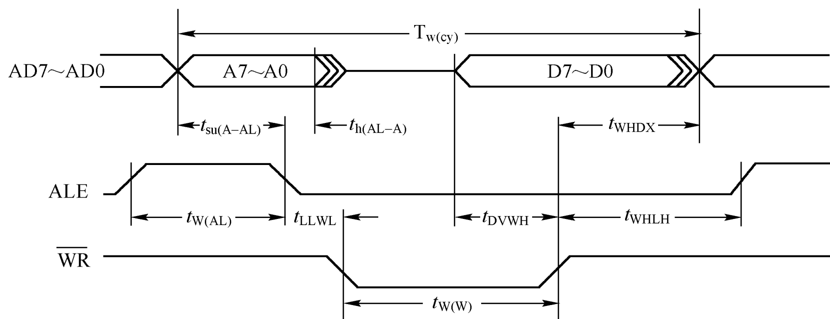


图 3-8　SJA1000 写周期时序图

表 3-50　与读写有关的信号

符　　号	引　　脚	说　　明
AD7～AD0	2，1，23～28	多路地址/数据总线
ALE	3	下降沿锁存地址
$\overline{\text{CS}}$	4	片选
$\overline{\text{RD}}$	5	读信号，上升沿读出数据
$\overline{\text{WR}}$	6	写信号，上升沿写入数据
$\overline{\text{INT}}$	16	中断开漏输出

3.7　基于 51 系列单片机的 CAN 智能节点设计

3.7.1　硬件设计

根据上节介绍的 SJA1000 的时序图，可以设计 CAN 智能通信节点硬件电路原理如图 3-9 所示，从图中可以看出，电路主要由三部分构成：微处理器 89C51、独立 CAN 控制器 SJA1000、CAN 总线驱动器 82C250 和高速光耦 6N137。微处理器 89C51 负责 SJA1000 的初始化，通过控制 SJA1000 实现数据的接收和发送等通信任务，CAN 总线驱动器 82C250 是 CAN 控制器与 CAN 总线的接口器件。

SJA1000 的 AD0～AD7 连接到 89C51 的 P0 口，$\overline{\text{CS}}$ 连接到 89C51 的 P2.7 口。P2.7 为 0 时，CPU 片外存储器地址可选中 SJA1000，CPU 通过这些地址可对 SJA1000 执行相应的读/写操作。SJA1000 的 $\overline{\text{RD}}$、$\overline{\text{WR}}$、ALE 分别与 89C51 的对应引脚相连，SJA1000 的 $\overline{\text{INT}}$ 接 89C51 的 $\overline{\text{INT0}}$，89C51 可以通过中断的方式访问 SJA1000。

为了增强 CAN 总线节点的抗干扰能力，SJA1000 的 TX0 和 RX0 并不是直接与 82C250 的 TXD 和 RXD 相连，而是通过高速光耦 6N137 与 82C250 相连，这样就很好地实现了总线上各 CAN 节点间的电气隔离，以实现保护 CAN 控制器的目的。光耦 6N137 是高速光隔离器，最高速度为 10Mbit/s。光耦部分电路所采用的两个电源 V_{CC} 和 V_{DD} 必须完全隔离，否则采用光耦就失去了意义。这虽然增加了接口电路的复杂性，但是却提高了节点的稳定性和安全性。

Rs 引脚可以控制 82C250 的工作状态。图 3-9 中，82C250 的 Rs 脚接一个电阻后再接地，用于控制上升和下降斜率，从而减小射频干扰。

82C250 与 CAN 总线的接口部分也采用了一定的安全和抗干扰措施。82C250 的 CANH 和 CANL 引脚各自通过一个 5Ω的电阻与 CAN 总线相连，电阻可起到一定的限流作用，保护 82C250 免受过流的冲击。CANH 和 CANL 与地之间分别并联了一个 30pF 的小电容，可以起到滤除总线上高频干扰的作用。另外，在两根 CAN 总线输入端与地之间分别接了一个瞬态抑制二极管，当两输入端与地之间出现瞬变干扰时，通过瞬态抑制二极管可以起到一定的保护作用。

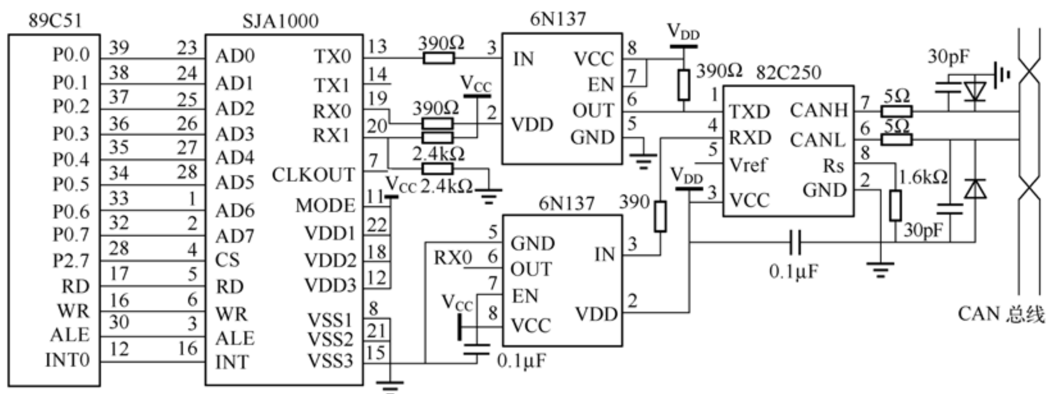


图 3-9　单片机与 CAN 总线接口电路

3.7.2　软件设计

CAN 总线智能节点的软件主要包括 SJA1000 的初始化、CAN 报文发送、CAN 报文接收 3 部分，本节将介绍基于 BasicCAN 模式下的软件设计，为便于读者阅读，在 51 系列单片机的 C 语言编程环境下编写程序。

1．SJA1000 初始化

SJA1000 初始化在复位模式下进行，主要包括工作方式设置、接收滤波方式设置、接收屏蔽寄存器设置和接收代码寄存器设置、波特率设置和中断允许寄存器设置等。

```
#define CR_CAN          XBYTE[0x7f00]
#define CMR_CAN         XBYTE[0x7f01]
#define SR_CAN          XBYTE[0x7f02]
#define IR_CAN          XBYTE[0x7f03]
#define ACR_CAN         XBYTE[0x7f04]
#define AMR_CAN         XBYTE[0x7f05]
#define BTR0_CAN        XBYTE[0x7f06]
#define BTR1_CAN        XBYTE[0x7f07]
#define OCR_CAN         XBYTE[0x7f08]
#define CDR_CAN         XBYTE[0x7f1f]
void initial_can(void)
{
  CR_CAN=0X01;                //SJA1000 进入复位模式，进行初始化；
  CDR_CAN=0X00;               //时钟分频寄存器选择 SJA1000 为 BasicCAN 模式；
  ACR_CAN=ACR_ID;             //验收代码寄存器设置；
  AMR_CAN=0xff;               //验收屏蔽寄存器设置；
  BTR0_CAN=0x47;
  BTR1_CAN=0x2f;              //总线定时寄存器 0 和寄存器设置波特率为 10kbit/s；
  OCR_CAN=0xDA;               //输出模式设为推挽模式；
  CMR_CAN=0x0C;               //命令寄存器清除数据溢出状态位和释放接收缓冲器；
  CR_CAN=0x72;                //控制寄存器使 SJA1000 回到操作模式；
}
```

2．CAN 报文发送

CAN 报文发送子程序主要完成 CAN 节点报文的发送。发送前，先做一些必要的判断。发送时，将待发送的数据按照特定格式组合成报文，然后送入 SJA1000 发送缓存，最后启动 SJA1000 的发送。

```
void send(void)
{
uchar i;
uchar xdata *sendbufpt;
if ( (SR_CAN&0X04) = = 0x04 )              //判断发送缓冲器状态位是否置位，是则发送；
{
sendbufpt=0x7f0a;                          //SJA1000 发送缓冲区首址；
sendbufpt++=D_ID;                          //发送报文 ID 送入发送缓冲区；
*sendbufpt++=0x01;                         //报文的长度为一个字节；
          for (i=0;i<8;i++)
          {
*sendbufpt++=senddata[i] ;
}        //逐个将报文发送到发送缓冲区；
     CMR_CAN=0x01;                         //启动 SJA1000 开始发送；
}
}
```

3．CAN 报文接收

CAN 报文接收子程序主要完成 CAN 节点报文的接收。接收子程序在接收报文的同时，还要处理总线关闭、错误报警、接收溢出等情况。SJA1000 的接收方法包括查询和中断两种方式，这里采用中断接收方式。

```
void receiveint(void) interrupt 0
{
uchar xdata *recbufpt;
  uchar i;
  EA=0;                          //89C51 关闭一切中断；
  IR_CAN=0x7f03;                 //SJA1000 接收缓冲区首地址；
  if (IR_CAN&0x01==0x01)         //判断接收缓冲器是否有数据，有则开始读取数据；
  {
recbufpt=0x7f16;
for (i=0; i<8; i++)
recdata[i]=*recbufpt;
}
else
{
CMR_CAN=0x08;                    //接收缓冲器无数据则清除数据溢出状态位；
}
  CMR_CAN=0x04;                  //释放接收缓冲区；
          EA=1;                  //89C51 开放所有中断；
}
```

智能的 CAN 通信节点在自主实现 CAN 报文的收发基础上，还可以根据应用需求完成一定的信号处理和控制功能，实现一个相对独立的通信节点单元。

3.8 本章小结

CAN 控制器主要由实现 CAN 总线协议和与微处理器接口的两部分电路组成。通过微处理器对它的编程，可以设置它的工作方式，控制它的工作状态，进行数据的发送和接收，从而把应用层建立在它的基础之上。由于 CAN 总线性能突出，现在 CAN 总线在很多领域的应用都得到迅速发展。这使得许多半导体器件厂商竞相推出各种 CAN 总线器件产品，其中主要以支持 CAN2.0A 协议的 Philips 公司的 PCA82C200 和支持 CAN2.0B 协议的 Philips 公司的 SJA1000 为代表。由于各个厂家的产品都严格按照已经制定的 CAN 规范和国际标准来做，所以只要精通一种 CAN 控制器，其余的都可以触类旁通。本章主要以 SJA1000 为对象，介绍 CAN 控制器的结构、功能及其应用。

思考题与习题

3.1 简要介绍 SJA1000 在 CAN 总线中的作用。

3.2 简要介绍 SJA1000 组成及其功能。

3.3 分析 SJA1000 的读写时序，解释 SJA1000 与 89C51 系列单片机的硬件接口。

3.4 假设 SJA1000 的晶体振荡频率为 16MHz，总线定时寄存器 0 和 1 参数分别为 0x1f 和 0xff，计算 CAN 通信的位速率。

3.5 对于上述设计的基于单片机的 CAN 总线通信节点，有 A、B 两个节点要求进行通过 CAN 总线进行数据通信，假定 SJA1000 的晶体振荡频率为 16MHz，数据通信的位速率为 100kbit/s，采用单滤波方式，请分别写出 A、B 两个单片机在扩展模式下的对应的初始化程序。

第 4 章　典型 CAN 总线驱动器

CAN 总线驱动器提供 CAN 总线控制器与物理总线之间的接口以及对 CAN 总线的差动发送和接收功能，是影响系统网络性能的重要器件。实际应用中，根据不同的场合要选择不同类型的 CAN 总线驱动器。

本章要点

- CAN 总线驱动器 PCA82C250 和 PCA82C251 主要特性、功能及应用介绍；
- 高速 CAN 驱动器 TJA1040 的主要特性、功能及应用介绍；
- 高速 CAN 驱动器 TJA1050 的主要特性、功能及应用介绍；
- 几种典型的 CAN 总线驱动器的比较。

4.1　CAN 总线驱动器概述

CAN 总线控制器的信息必须经过总线才能进行数据的传输。由于 CAN 总线支持的节点的数量多，而且每个节点的电气特性有差异，如果直接将 CAN 总线控制器挂接到总线上，将造成不可预测的后果。另外，由于 CAN 总线控制器给出的是固定的数字信号，为了灵活应用，一般在 CAN 控制器与 CAN 物理总线之间加一个 CAN 总线驱动器。

CAN 总线驱动器分为双线驱动器和单线驱动器。所谓的单线驱动器，亦即只有一条数据通信总线，所有信号的发送和接收均由这条总线实现，但是总线上所有挂接的 CAN 通信节点均需共地。双线驱动器是应用最为广泛的一种驱动器，该类驱动器对总线提供差动发送能力，对 CAN 控制器提供差动接收能力。

目前，双线 CAN 驱动器已经广泛地应用在高速通信和中速通信的应用系统中，并以追求传输速度为终极目标。双线 CAN 驱动器由于采用了双绞线，大大降低了内部噪声，从而也使得高速通信得以实现。但在实际的应用中，特别是对一些机械装置的控制，以及一些传感器数据的读操作，其频率一般不高，从而其对数据传输的速度要求大大降低。一般在小于 50kbit/s 的数据传输速率条件下，采用双线驱动器相对于采用单线驱动器将会提高成本。而单线驱动器作为一个具有很高性价比的网络方案，将在一些对成本敏感的领域有着广泛的应用前景。

4.2　CAN 总线驱动器 PCA82C250/251

CAN 总线驱动器 PCA82C250 和 PCA82C251 提供 CAN 控制器与物理总线之间的接口。它最初是为汽车中的高速应用（达 1Mbit/s）而设计的，可提供对总线的差动发送和接收功能。

4.2.1 PCA82C250/251 的主要特性

CAN 总线驱动器 PCA82C250 和 PCA82C251 是先进的驱动器产品，其主要特性如下：

① 与 ISO-11898 标准完全兼容；

② 高传输速率（最高可达 1Mbit/s）；

③ 具有抗汽车环境下的瞬间干扰，保护总线的能力；

④ 采用斜率控制（Slope Control），降低射频干扰（RFI）；

⑤ 过热保护；

⑥ 总线与电源及地之间的短路保护；

⑦ 低电流待机模式；

⑧ 未上电节点不会干扰总线；

⑨ 总线至少可连接 110 个节点。

4.2.2 PCA82C250/251 的基本性能

1．PCA82C250 与 PCA82C251 的主要差异

驱动器是协议控制器和物理传输线路之间的接口，可以对总线提供差动发送能力，对 CAN 控制器提供差动接收能力。如 ISO-11898 标准中所述，PCA82C250/251 驱动器可以用高达 1Mbit/s 的位速率在两条有差动电压的总线电缆上传输数据。这两个器件都可以在额定电源电压分别是 12V（PCA82C250）和 24V（PCA82C251）的 CAN 总线系统中使用。

PCA82C250 和 PCA82C251 的功能相同，还可在同一网络中互相通信，且它们的引脚和功能兼容，也就是说它们可以用在相同的印制电路板上。表 4-1 列出了两个器件的主要不同。

表 4-1 PCA82C250 和 PCA82C251 驱动器的主要差异

	PCA82C250	PCA82C251
系统额定电源电压	12V	12V 和/或 24V
最大的总线终端 DC 电压($0V < V_{CC} < 5.5V$)	$8V < V_{CANL,H} < +18V$	$-40V < V_{CANL,H} < +40V$
最大的瞬间总线终端电压（ISO 7637）	$-150V < V_{tr} < +100V$	$-200V < V_{tr} < +200V$
扩展扇出应用时最小驱动器电源电压（ $R_L = 45\Omega$ ）	$V_{CC} > 4.9V$	$V_{CC} > 4.5V$

由于 PCA82C251 有更高的击穿电压，还可以在这个电源电压范围内驱动低至 45Ω的总线负载，所以建议在普通的工业应用中使用这个器件。而且 PCA82C251 在隐性状态下的上拉电流更小，这使得在掉电情况下总线的输出特性有一定改善。

2．PCA82C250 的基本性能

82C250 驱动器输出的额定总线电平，如图 4-1 所示，其功能框图如图 4-2 所示，其性能参数和引脚功能见表 4-2 和表 4-3。

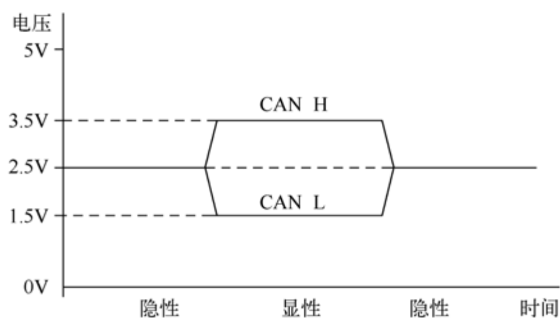


图 4-1　根据 ISO 11898 驱动器输出的额定总线电平

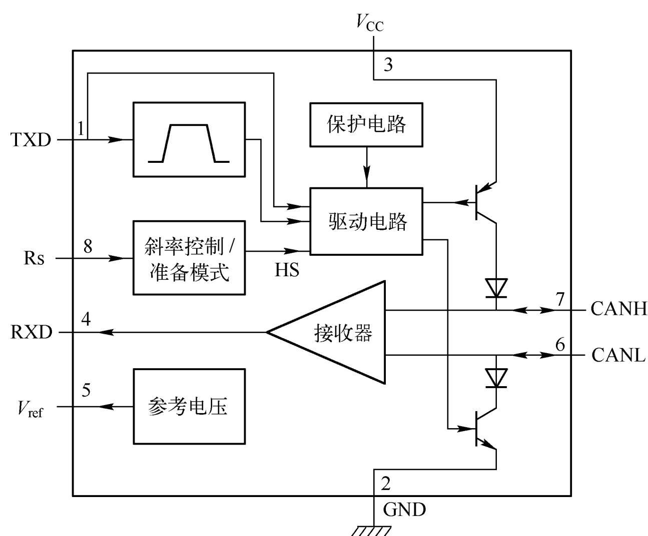


图 4-2　PCA82C250 的功能图

表 4-2　PCA82C250 的基本性能参数

符　号	参　数	条　件	最 小 值	典 型 值	最 大 值	单　位
V_{CC}	电源电压		4.5	—	5.5	V
I_{CC}	电源电流	显性位，$V1$=1V	—	—	70	mA
		隐性位，$V1$=4V	—	—	14	mA
		待机模式	—	100	170	μA
V_{CAN}	CANH，CANL 脚直流电压	$0V < V_{CC} < 5.5V$	−8	—	18	V
ΔV	差动总线电压	$V1$=1V	1.5	—	3.0	V
$V_{diff(z)}$	差动输入电压（显性位）	非待机模式	−1.0	—	0.4	V
$V_{diff(d)}$	差动输入电压（隐性位）	非待机模式	1.0	—	5.0	V
Y_d	传播延迟	高速模式	—	—	50	ns
T_{amb}	工作环境温度		−40	—	125	℃

表 4-3　PCA82C250 的引脚功能

标　记	引　脚	功能描述
TXD	1	发送数据输入
GND	2	接地
VCC	3	电源
RXD	4	接收数据输出
Vref	5	参考电压输出
CANL	6	低电平 CAN 电压输入/输出
CANH	7	高电平 CAN 电压输入/输出
Rs	8	斜率电阻输入

4.2.3　PCA82C250/251 的功能描述

下面对 PCA82C250/251 的功能模式进行分析。PCA82C250/251 三种不同的工作模式，分别是高速模式、斜率模式和准备模式。模式控制通过 Rs 控制引脚完成，如图 4-3 所示。

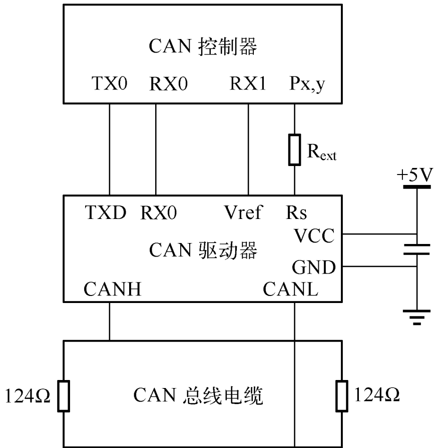


图 4-3　PCA82C250/251 驱动器的功能接线

$P_{x,y}$ 为高时，PCA82C250/251 切换到准备模式（$V_{Rs}>0.75V_{CC}$）。

$P_{x,y}$ 为低时，PCA82C250/251 切换到普通工作模式。普通工作模式可以是高速模式或低斜率模式，这由连接到 Rs 的电阻决定。

其中 $0\Omega < R_{ext} < 1.8k\Omega$时为高速模式（$V_{Rs}<0.3\ V_{CC}$）；$16.5k\Omega < R_{ext} < 140k\Omega$时为斜率控制模式（$10\mu A<-I_{Rs}<200\mu A$）。

1．高速模式

在这个模式中适合执行最大的位速率或最大的总线长度。这种模式的总线输出信号采用尽可能快的速度切换，因此一般使用屏蔽的总线电缆来防止可能的扰动。

高速模式通过$V_{Rs}<0.3V_{CC}$ 来选择，将 Rs 控制输入直接连接到微控制器的输出口或者低电平或者一个高电平有效的复位信号（如图 4-3 所示），就可以实现。

高速模式中，驱动器有效的循环延迟时间可以低至 145ns（当 $T_{amb}>85$℃时，是

155ns）。根据 CAN 位定时的要求，有效的循环延迟是显性边沿循环延迟以及显性和隐性边沿循环延迟的平均值之中的最大值。

$$t_{\text{loop.eff}} = \max\{0.5 \times (t_{\text{onRXD}} + t_{\text{offRXD}}), t_{\text{onRXD}}\}$$

2．斜率控制模式

在一些应用中由于考虑到系统的成本等问题而使用非屏蔽的总线电缆。但是，使用非屏蔽电缆意味着驱动器要满足额外的要求，譬如：电磁兼容性（EMC）问题。如果使用非屏蔽总线电缆，PCA82C250/251 的总线信号转换速度应有意降低。转换速度可以通过连接在控制引脚 Rs 上的串联阻抗值 R_{ext} 来调整，根据 CAN 的位定时要求，转换速度下降将增加总线节点的循环延迟，因此在给定的位速率下，总线长度减少，或者说在给定的总线长度下，位速率降低。斜率控制模式中，总线输出的转换速度大致和流出引脚 Rs 的电流成比例，电流范围在 $10\mu A < -I_{\text{Rs}} < 200\mu A$ 之间。如果 Rs 引脚的输出电流在这个范围中，引脚 Rs 将输出大约 $0.5V_{\text{CC}}$ 的电压。当在 Rs 引脚和地电平之间应用一个适当的电阻时，驱动器被设置成斜率控制模式。单凭经验来说，这个电阻阻值要在范围 $16.5k\Omega < R_{\text{ext}} < 140k\Omega$ 之间才符合上述的 Rs 输出电流范围。

合适的 R_{ext} 取值范围可以用下面的斜率控制模式的界限值计算：

$10\mu A < -I_{\text{Rs}} < 200\mu A$ 和 $0.4V_{\text{CC}} < V_{\text{Rs}} < 0.6V_{\text{CC}}$。

则 R_{ext} 的最小和最大值可用下面关系式算出：

$$R_{\text{ext}} > \frac{0.6V_{\text{CC.max}}}{I_{\text{Rs.max}}} = \frac{0.6V_{\text{CC.max}}}{200\mu A}$$

$$R_{\text{ext}} < \frac{0.4V_{\text{CC.max}} - V_{\text{OL.max}}}{I_{\text{Rs.min}}} = \frac{0.4V_{\text{CC.max}} - V_{\text{OL.max}}}{10\mu A}$$

3．准备模式

准备模式在需要将功率消耗（譬如是暂时性的）降到最低时使用。当 $V_{\text{Rs}} > 0.75V_{\text{CC}}$ 时，驱动器进入准备模式。

系统的功耗在准备模式可被有效降低。这个模式基本上用于电池供电的应用中，例如：汽车停车的时候。要进入准备模式，驱动器的控制输入 Rs 上要加一个逻辑高电平。这可以通过直接将一个输出端口引脚连接到 Rs 或通过任何合适的斜率控制电阻 R_{ext} 来实现。准备模式中，发送器的功能和接收器的输入偏置网络都关断，以减少功率消耗。参考电压输出和基本的接收器功能仍然活动，但以非常低的功耗工作。如果在总线上传输一个报文，系统可被重新激活。在检测 3μs 长的显性总线电平后，驱动器将通过 RXD 向协议控制器输出一个唤醒中断信号。在检测到 RXD 的下降沿后，控制器把 Rs 引脚置为逻辑低电平，这样驱动器就可以切换到普通传输模式。由于在准备模式中工作速度缓慢，驱动器回到普通接收速度取决于逻辑的延迟时间（Rs 的下降沿）。在总线速度很高的情况下，驱动器在准备模式（如 Rs 引脚仍然为高电平）中不可能正确地接收报文。

4.2.4 PCA82C250/251 的典型应用

在 PCA82C250/251 驱动器的典型应用中，首先分析没有实现光电隔离的普通应用。如

图 4-3 所示，CAN 控制器通过串行数据输出线 TX0 和串行数据输入线 RX0 连接到驱动器，驱动器通过有差动发送和接收功能的两个总线终端 CANH 和 CANL 连接到总线电缆，输入 Rs 用于模式控制，参考电压输出 V_{ref} 的输出电压是额定 V_{CC} 的 0.5 倍，其中驱动器的额定电源电压是 5V。总线控制器输出一个串行的发送数据流到驱动器的 TXD，引脚内部的上拉功能将 TXD 输入设置成逻辑高电平。在隐性状态中，CANH 和 CANL 输入通过典型内部阻抗是 17kΩ的接收器输入网络，偏置到 2.5V 的额定电压。另外，如果 TXD 是逻辑低电平，总线的输出级将被激活，在总线电缆上产生一个显性的信号电平，输出驱动器由一个源输出级和一个下拉输出级组成，CANH 连接到源输出级，CANL 连接到下拉输出级，在显性状态中 CANH 的额定电压是 3.5V，CANL 是 1.5V。

图 4-4 是 PCA82C250/251 的应用中有光电隔离的实例。注意在实际应用中，如果位速率很高，例如高于 500kbit/s，则应考虑使用延迟小于 40ns 的高速光耦。

如果没有一个总线节点传输一个显性位，总线处于隐性状态，即网络中所有 TXD 输入是逻辑高电平。另外，如果一个或更多的总线节点传输一个显性位，即至少一个 TXD 输入是逻辑低电平，则总线从隐性状态进入显性状态，即具有逻辑线与功能。

接收器的比较器将差动的总线信号转换成逻辑信号电平，并在 RXD 输出，接收到的串行数据流传送到总线协议控制器译码。接收器的比较器总是活动的，也就是说当总线节点传输一个报文时，它同时也监控总线，这就要求有诸如安全性和支持非破坏性逐位竞争等 CAN 策略。一些控制器提供一个模拟的接收接口（RX0，RX1），RX0 一般需要连接到 RXD 输出，RX1 需要偏置到一个相应的电压电平，这可以通过 Vref 输出或一个电阻电压分配器实现。

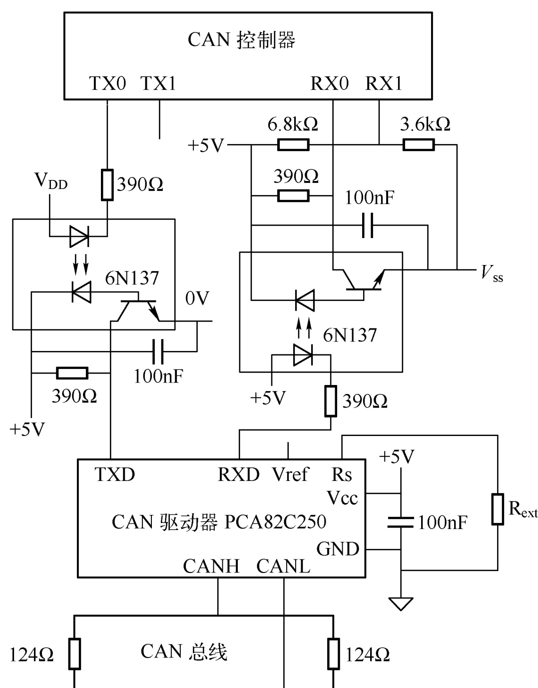


图 4-4　带光电隔离的驱动器应用

在图 4-3 中，驱动器直接连接到协议控制器及其应用电路。如果需要电流隔离，光耦合器可以如图 4-4 所示的一样，放置在驱动器和协议控制器之间。使用光耦合器时要注意选择正确的默认状态，特别是在隔开的协议控制器电路一边没有上电时，这种情况下连接到 TXD 的光耦合器应该是“暗”的，即 LED 关断。当光耦合器是断开（暗）时，驱动器的 TXD 输入是逻辑高电平，可以达到自动防故障的目的。

然而在总线控制器和驱动器之间使用光耦合器，通常会增加总线节点的循环延迟。信号在每个节点要从发送和接收路径通过这些器件两次，这将减少在给定位速率条件下可使用的最大的总线长度。这在计算由于 CAN 网络中的传播延迟而造成对可以使用的最大总线长度被限制时要考虑。

4.3 高速 CAN 总线驱动器 TJA1040

TJA1040 是控制器局域网（CAN）协议控制器连接物理总线的高速驱动器。作为 PCA82C250/251 的后继产品，它的引脚不仅和 PCA82C250 一致，而且在不上电状态下有理想的无源性能，支持远程唤醒。

4.3.1 TJA1040 的主要特性

从使用者的角度分析，TJA1040 具有较为优良的特性，主要包括以下几点：

① 完全符合 ISO 11898 标准；
② 速度高达 1Mbit/s；
③ 电磁辐射 EME 非常低；
④ 差动接收器具有较宽的共模范围，可抗电磁干扰 EMI；
⑤ 处于不上电状态的驱动器会从总线脱离；
⑥ 输入级符合 3.3V 和 5V 的器件；
⑦ 如果使用分裂终端电压源可以稳定隐性总线电平，进一步降低 EME；
⑧ 至少可以连接 110 个节点；
⑨ 消耗电流极低的待机模式具有通过总线唤醒远程的功能；
⑩ 发送数据 TXD 显性超时功能；
⑪ 在汽车的瞬态环境下对总线引脚进行保护；
⑫ 防止总线引脚和引脚 SPLIT 对电池和对地短路；
⑬ 热保护。

4.3.2 TJA1040 的基本性能

TJA1040 主要应用在客车的高速应用上，速度可达 1Mbit/s。TJA1040 为总线提供差动的发送功能，为 CAN 控制器提供差动的接收功能。

在引脚和功能上，TJA1040 是 PCA82C250/251 高速 CAN 驱动器的后继产品。而且，它的引脚和 82C250 一致。TJA1040 有优秀的 EMC 性能，而且在不上电状态下有理想的无源性能，它还提供低功耗管理，支持远程唤醒。TJA1040 功能框图，如图 4-5 所示。其性能参数和引脚功能见表 4-4 和表 4-5。

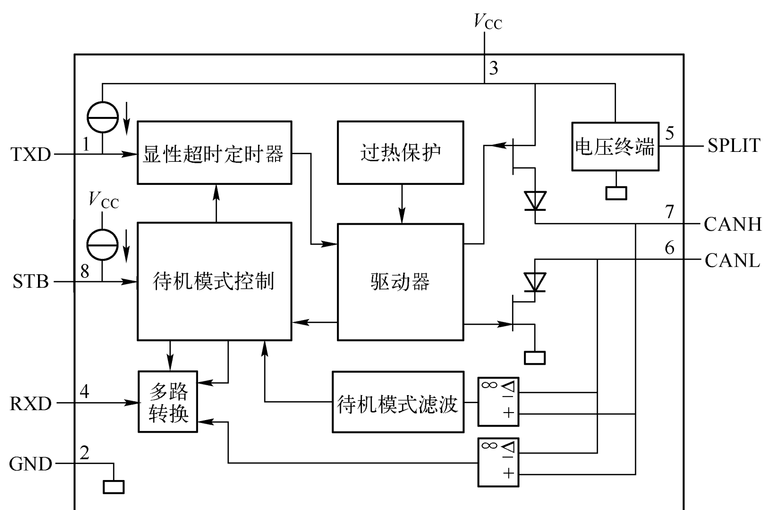


图 4-5　TJA1040 结构图

表 4-4　TJA1040 快速参考数据

助 记 符	参　　数	条　　件	最 小 值	最 大 值	单　　位
V_{CC}	电源电压		4.75	5.25	V
I_{CC}	电源电流	待机模式	5	15	μA
V_{CANH}	引脚 CANH 的直流电压	$0 < V_{CC} < 5.25V$,无时间限制	−27	40	V
V_{CANL}	引脚 CANL 的直流电压	$0 < V_{CC} < 5.25V$,无时间限制	−27	40	V
V_{SPLIT}	引脚 SPLIT 的直流电压	$0 < V_{CC} < 5.25V$,无时间限制	−27	40	V
T_{vj}	实际连接点温度		−40	150	℃
$V_{esd(HBM)}$	所有引脚的静电放电电压	人体模型（HBM）	−4	4	V
$t_{PD(TXD-RXD)}$	TXD 到 RXD 的传播延迟	$V_{SB} = 0V$	—	255	ns

表 4-5　TJA1040 引脚描述

标　　记	引　　脚	功 能 描 述
TXD	1	发送数据输入
GND	2	接地
VCC	3	电源电压
RXD	4	接收数据输出(从总线读出数据)
SPLIT	5	共模稳压输出
CANL	6	低电平 CAN 总线
CANH	7	高电平 CAN 总线
STB	8	待机模式控制输入

4.3.3　TJA1040 的功能描述

1．工作模式

TJA1040 有正常和待机两种工作模式，可以通过引脚 STB 选择。表 4−6 对这两种模式分别作了描述。

表 4−6　TJA1040 工作模式

模　　式	引脚 STB	引脚 RXD	
		低	高
正常模式	L	总线显性	总线隐性
待机模式	H	检测到唤醒请求	没有检测到唤醒请求

（1）正常模式

在这个模式中，驱动器可以通过总线 CANH 和 CANL 发送和接收数据，见图 4−5 的结构图。差动接收器将总线上的模拟数据，转换成数字数据通过多路转换器（MUX）输出到 RXD。总线线路上输出信号的斜率是固定的并进行了优化，保证有很低的电磁辐射（EME）。

（2）待机模式

在这种模式中，发送器和接收器都关断，只用低功耗的差动接收器监控总线。V_{CC} 上的电源电流减少到最小，但仍保证抗电磁干扰的性能，并能识别出总线上的唤醒事件。

在这种模式中，总线两端都接到地，将电源电流（I_{CC}）减到最小。在 RXD 的高端驱动器（high-side driver）上串联一个二极管，防止不上电状态下有反向电流从 RXD 流向 V_{CC}。在正常模式中，这个二极管被旁路，但它在待机模式中可以减少电流的消耗，所以没有被旁路。

2．分解网络

分解网络（split circuit）是一个 $0.5V_{CC}$ 的直流稳压源。它只在正常模式中接通。待机模式时，引脚 SPLIT 悬空。分解网络可以通过将引脚 SPLIT 连接到分裂终端的中心抽头，来稳定隐性共模电压，如图 4−6 所示。如果由于在网路中存在不上电的驱动器，它们在总线和地之间有显著的漏电流使隐性总线电压小于 $0.5V_{CC}$，分解网络会将这个隐性电压稳定为 $0.5V_{CC}$。因此启动发送时不会在共模信号上产生阶跃，从而保证电磁辐射（EME）性能。

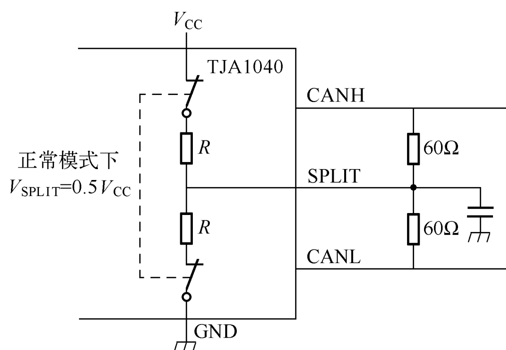


图 4−6　稳压电路举例

3．唤醒

在待机模式中，总线由低功耗的差动比较器监控。一旦低功耗的差动比较器检测到一个持续时间大于总线周期的显性总线电平，引脚 RXD 变低电平。

4．过热检测

输出驱动器在过热时会受到保护。如果实际连接点温度超过了 165℃，输出驱动器会被禁止，直到实际连接点温度低于典型的 165℃后，TXD 才会再一次变成隐性。因此输出驱动器的振幅不会受到温度漂移的影响。

5．显性超时功能

当引脚 TXD 由于硬件或软件程序的错误而被持续地置为低（电平），“TXD 显性超时”定时器电路可以防止总线进入持续的显性状态（阻塞所有网络通信）。这个定时器是由引脚 TXD 的负跳沿触发。如果引脚 TXD 的低电平持续时间超过内部定时器的值（t_{dom}），驱动器会被禁止，强制使总线进入隐性状态。定时器用引脚 TXD 的正跳沿复位。TXD 显性超时时间（t_{dom}）定义了允许的最小位速率是 40kBaud。

6．自动防故障功能

引脚 TXD 提供了一个向 V_{CC} 的上拉，使引脚 TXD 在不使用时保持隐性电平。

引脚 STB 提供了一个向 V_{CC} 的上拉，当不使用引脚 STB 时使驱动器进入待机模式。

如果 V_{CC} 掉电，引脚 TXD、STB 和 RXD 会变成悬空状态，以防止通过这些引脚产生反向电流。

4.3.4 TJA1040 的典型应用

图 4-7 显示了如何将 TJA1040 集成到应用中。这个应用例子中，TJA1040 和微控制器共用一个 5V 稳压器。TJA1040 除了连接 V_{CC} 电源外，还直接连接电池电源，确保在睡眠模式中关断 V_{CC} 电源后 TJA1040 仍然有本地和远程唤醒能力。

TJA1040 通过 3 条信号线连接到主控制器，微控制器通过信号 STB 控制 TJA1040 的工作模式。引脚 STB 提供内部的下拉电流，如果它没有连接收发器，就会进入待机模式。TXD 和 RXD 分别代表发送和接收位流。为了改善系统的 EMC 性能，微控制器和 TJA1040 之间的接口线可以选择串联大约 1kΩ的电阻，注意这些串联电阻会轻微地增加传播延迟。TJA1040 相关的总线引脚是两个总线终端 CANH 和 CANL 以及引脚 SPLIT。SPLIT 可被连接到分裂终端的中心抽头，为共模电压提供直流稳定性或者直接让引脚开路。

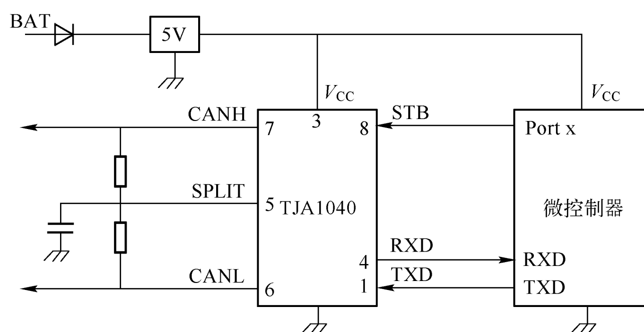


图 4-7　TJA1040 典型应用电路

4.4　高速 CAN 总线驱动器 TJA1050

TJA1050 是 CAN 驱动器 PCA82C250 的后继产品。TJA1050 不提供待机模式，特别是在不上电环境下具有无源特性。由于采用了最新的 EMC 技术，TJA1050 比 PCAC250/251 的抗电磁干扰性能提高了 20dB。

4.4.1　TJA1050 的主要特性

首先介绍 TJA1050 的主要电气特性，主要有以下几点：

① 与“ISO 11898”标准完全兼容；

② 速度高（最高可达 1Mbit/s）；

③ 低电磁辐射（EME）；

④ 具有宽带输入范围的差动接收器，可抗电磁干扰（EMI）；

⑤ 没有上电的节点不会对总线造成干扰；

⑥ 发送数据（TXD）控制超时功能；

⑦ 发送禁止时的静音模式；

⑧ 在暂态时自动对总线引脚进行保护；

⑨ 输入级与 3.3V 装置兼容；

⑩ 热保护；

⑪ 对电源和地的防短路功能；

⑫ 可以连接至少 110 个节点。

4.4.2　TJA1050 的基本性能

TJA1050 是 CAN 协议控制器和物理总线之间的接口。TJA1050 可以为总线提供不同的发送性能，为 CAN 控制器提供不同的接收性能。TJA1050 是 PCA82C250 高速 CAN 驱动器的后继产品，引脚和参数具体见表 4-7 和表 4-8，内部结构图（见图 4-8）。TJA1050 在以下方面作了重要的改进：

① CANH 和 CANL 理想配合，使电磁辐射减到更低；

② 在有不上电节点时，性能有所改进。

表 4-7　TJA1050 引脚描述

标　　记	引　　脚	功能描述
TXD	1	发送数据输入
GND	2	接地
VCC	3	电源电压
RXD	4	接收数据输出(从总线读出数据)
Vref	5	参考电压输出
CANL	6	低电平 CAN 总线
CANH	7	高电平 CAN 总线
S	8	待机模式控制输入

表 4-8　TJA1050 快速参考数据

助　记　符	参　　数	条　　件	最　小　值	最　大　值	单　　位
V_{CC}	电源电压		4.75	5.25	V
V_{CANH}	引脚 CANH 的直流电压	$0 < V_{CC} < 5.25V$,无时间限制	−27	40	V
V_{CANL}	引脚 CANL 的直流电压	$0 < V_{CC} < 5.25V$,无时间限制	−27	40	V
$V_{i(dif)(bus)}$	不同的总线输入电压	控制	1.5	3	V
$t_{PD(TXD-RXD)}$	TXD 到 RXD 的传播延迟	$V_S = 0V$	—	250	ns
T_{amb}	环境温度		−40	125	℃

4.4.3　TJA1050 的功能描述

TJA1050 是 CAN 协议控制器和物理总线之间的接口。它最初是应用在波特率范围在 60k Baud 到 1M Baud 的高速自动化应用中。TJA1050 可以为总线提供不同的发送性能，为 CAN 控制器提供不同的接收性能，而且它与“ISO-11898”标准完全兼容。

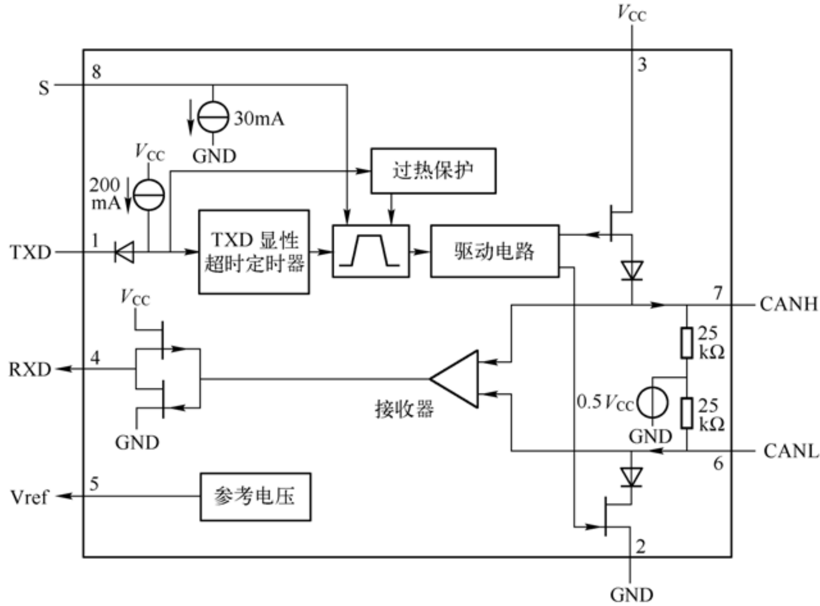


图 4-8　TJA1050 结构图

TJA1050 功能见表 4-9。TJA1050 有一个电流限制电路，保护发送器的输出级，使由正或负电源电压意外造成的短路不会对 TJA1050 造成损坏。

表 4-9　TJA1050 功能表

V_{CC}	TXD	S	CANH	CANL	总 线 状 态	RXD
4.75～5.25	0	0(或悬空)	高	低	控制	0
4.75～5.25	×	×	$0.5V_{CC}$	$0.5V_{CC}$	隐性	1
4.75～5.25	1(或悬空)	×	$0.5V_{CC}$	$0.5V_{CC}$	隐性	1
<2V(不加电)	×	×	$0V < V_{CANH} < V_{CC}$	$0V < V_{CANL} < V_{CC}$	隐性	×
$2V < V_{CC} < 4.75V$	>2V	×	$0V < V_{CANH} < V_{CC}$	$0V < V_{CANL} < V_{CC}$	隐性	×

TJA1050 还有一个温度保护电路，当与发送器的连接点的温度超过大约 165℃时，会断开与发送器的连接。因为发送器消耗了大部分的功率，所以这个集成电路的功率消耗和温度会较低。但是此时 IC 的其他功能仍继续工作。当引脚 TXD 变高（电平），发送器由关闭状态复位。当总线短路时，尤其需要这个温度保护电路。

在通电的瞬间，引脚 CANH 和 CANL 也受到保护（根据“ISO 7637”）。通过引脚 S 可以选择高速模式或静音模式这两种工作模式。

高速模式就是普通的工作模式，将引脚 S 接地可以进入这种模式。如果引脚 S 没有连接，高速模式就是默认的工作模式。

在静音模式中，发送器是禁止的。但 IC 的其他功能可以继续使用。将 S 引脚连接到 V_{CC} 可以进入这个模式。静音模式可以防止在 CAN 控制器不受控制时对网络通信造成堵塞。

当引脚 TXD 由于硬件或软件程序的错误而持久地为低（电平）时，“TXD 控制超时”定时器电路可以防止总线进入这种持久的支配状态（阻塞所有网络通信）。这个定时器是由引脚 TXD 的负跳变边缘触发。如果引脚 TXD 的低电平持续时间超过内部定时器的值，发送器会被禁止，使总线进入隐性状态。定时器由引脚 TXD 的正跳变边沿复位。

4.4.4 TJA1050 的典型应用

CAN 高速收发器的一般应用显示在图 4-9 中。其中协议控制器通过一条串行数据输出线 TXD 和一条串行数据输入线 RXD 连接到收发器，而收发器则通过它的两个有差动接收和发送能力的总线终端 CANH 和 CANL 连接到总线线路。它的引脚 S 用于模式控制，参考输出电压 *V*ref 提供一个 $V_{CC}/2$ 的额定输出电压，这个电压是作为带有模拟 Rx 输入的 CAN 控制器的参考电平。由于 SJA1000 具有数字输入，因此它不需要这个电压。收发器使用 5V 的额定电源电压。

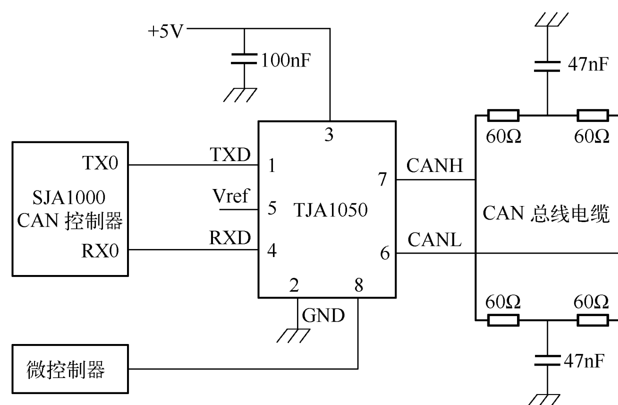


图 4-9　TJA1050 典型应用电路

协议控制器向收发器的 TXD 引脚输出一个串行的数据流，收发器的内部上拉功能将 TXD 引脚置为逻辑高电平，即总线输出驱动器在开路时是无源的。在隐性状态中，CANH 和 CANL

输入通过典型内部阻抗为25kΩ的接收器连接入网络，偏置到$V_{CC}/2$的电平电压。另外如果TXD是逻辑低电平将激活总线的输出级，并在总线上产生一个显性信号电平。输出驱动CANH由V_{CC}提供一个源输出，而CANL则向GND提供一个下拉输出。

如果没有总线节点发送一个显性位，则总线处于隐性状态；如果一个或多个总线节点发送一个显性位，总线就会覆盖隐性状态而进入显性状态（线与特性）。

接收器比较器将差动的总线信号转换成逻辑电平信号，并在RXD输出。总线协议控制器将接收到的串行数据流译码，接收器比较器总是激活的，即当总线节点发送一个报文时，它同时监控总线。这个功能可以用于支持CAN 的非破坏性逐位仲裁策略。

典型的总线采用一对双绞线，考虑到ISO 11898中定义的线性拓扑结构，总线两端都端接一个120Ω的额定电阻。这就要求总线额定负载是60Ω。终端电阻和电缆阻抗的紧密匹配确保了数据信号不会在总线的两端反射。

4.5 几种典型的CAN总线驱动器的比较

这里主要对PCA82C250/251、TJA1040、TJA1050等几种典型的CAN高速驱动器作一比较，包括它们之间的联系、区别、优缺点等。TJA1040/TJA1050与PCA82C250/251一样，是一个遵从ISO 11898的高速CAN驱动器，可以在汽车和工厂现场控制中使用。

TJA1050的设计使用了最新的EMC技术。它采用了先进的绝缘硅（Silicon-on-Insulator，SOI）技术进行处理。这样，TJA1050比PCA82C250/251（使用分离终端）的抗电磁干扰性提高了20dB。TJA1050集中在典型的“clamp-15”应用上使用，在汽车点火之后仍然保持不上电状态。因此TJA1050不提供待机模式。特别要注意的是它在不上电环境下的无源特性。

TJA1040是以TJA1050的设计为基础。由于使用了相同的SOI技术，TJA1040具有和TJA1050一样出色的EMC特性。和TJA1050不同的是，TJA1040像PCA82C250/251一样有待机模式，可以通过总线远程唤醒。这样，TJA1040可以认为是PCA82C250/251 的功能上的后继产品。TJA1040 还具有和 PCA82C250/251 一样的引脚和功能，所以 TJA1040 可以与PCA82C250/251兼容，并简单地替代PCA82C250/251。特别是TJA1040还首次提供在不上电环境下理想的无源特性。

TJA1040比PCA82C250/251有几个优势的地方：

如果不上电，则在总线上完全无源（如果V_{CC}关闭，则从总线上无法发现）；

在待机模式时，电流消耗非常低（最大15μA）；

改良的电磁辐射（EME）性能；

改良的电磁抗干扰（EMI）性能；

“SPLIT”引脚（代替Vref引脚，对总线的DC稳压很有效）。

TJA1040可以向下兼容PCA82C250/251，并且可以在很多已有的PCA82C250/251应用中使用，而硬件和软件不需要作任何修改。

4.5.1 应用方面的区别

PCA82C250/251、TJA1050 和 TJA1040 之间的区别见表 4-10。

表 4-10 PCA82C250/251、TJA1050 和 TJA1040 之间的区别

特 征	PCA82C250	PCA82C251	TJA1050	TJA1040
电源电压范围	4.5～5.5V	4.5～5.5V	4.75～5.25V	4.75～5.25V
总线引脚 6、7 的最大直流电压	−8～+18V	−36～+36V	−27～+40V	−27～+40V
循环延迟（TXD 至 RXD）（显性至隐性）	（Rs＝0）190ns；（Rs＝24kΩ）320ns	（Rs＝0）190ns	250ns	255ns
斜率控制	<170μA	<275μA	不支持	<15μA
有远程唤醒功能的待机模式	可变	可变	经过 EMC 优化的	
没上电的无源特性（V_{CC}＝0V 时总线引脚的漏电流）	<1mA（$V_{CANHL}=7V$）	<2mA（$V_{CANHL}=7V$）	<250μA（$V_{CANHL}=5V$）	<0μA（$V_{CANHL}=7V$）
共模电压的直流稳压	无	无	无	有

4.5.2 引脚的区别

图 4-10 是 PCA82C250/251、TJA1050 和 TJA1040 的引脚。除了两个重新命名的引脚外，这 3 个驱动器引脚是相同的。

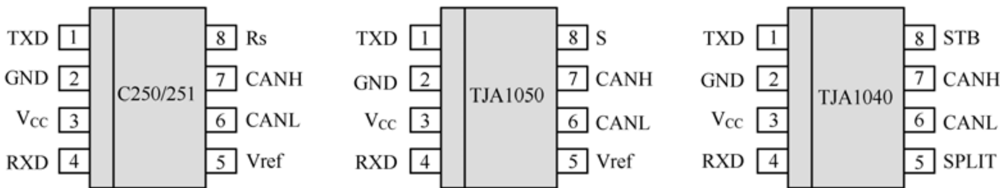


图 4-10 PCA82C250/251、TJA1050 和 TJA1040 的引脚配置

（1）模式控制引脚（引脚 8）

驱动器的引脚 8 用于控制驱动器的工作模式。这个引脚在 TJA1040 上的助记符是“STB”，是指待机模式；在 PCA82C250/251 上的助记符是“Rs”，是指斜率控制电阻；在 TJA1050 上的助记符是“S”，是指静音模式。虽然它们有不同的助记符，但模式控制是相同的，也就是说，普通模式或高速模式是通过在引脚 8 置低电平进入的。如果将这个引脚置高电平，驱动器会进入待机（PCA82C250/251、TJA1040）或静音模式（TJA1050）。

（2）参考电压引脚（引脚 5）

驱动器的引脚 5 提供了一个 $V_{CC}/2$ 的输出电压。PCA82C250/251 和 TJA1050 引脚 5 的助记符是“Vref”。引脚“Vref”是为了给前面 CAN 控制器的模拟比较器提供一个参考电压，使比较器能够准确地读出总线上的位值。

现在的 CAN 控制器通常有一个 RXD 信号的数字式输入，引脚 Vref 使用得越来越少了。

TJA1040 引脚 5 的助记符是“SPLIT”，它提供了 $V_{CC}/2$ 的电压。这个电源相关的低阻抗（典型值 600Ω）可以将共模电压稳定到额定的 $V_{CC}/2$，所以引脚“SPLIT”要被连接到分离终端的中间分接头。这样，即使由于未上电节点造成从总线到 GND 有很大的漏电流，共模电压

仍能够维持在接近额定值的 $V_{CC}/2$。

4.5.3 工作的模式区别

如前面所说驱动器的工作模式是由引脚 8 控制。表 4−11 显示了相关工作模式及其特征和相应的引脚 8 的设置。

表 4−11 相关工作模式以及提供的功能和引脚 8 相应的设置

工作模式	工作模式的特征	引脚 8 的信号电平		
		TJA1040	PCA82C250/251	TJA1050
正常（高速）	-发送功能 -接收功能	低	低或不连接	低或不连接
待机模式	-减少电流 -远程唤醒 -“Babbling Idiot”保护	高或不连接	高	———
斜率控制	可变斜率	———	通过 10kΩ<Rs<180kΩ 连接到 GND	———
静音	-“Babbling Idiot”保护 -“只接收特性”	———	———	高

下面对不同的工作模式和所提供的功能进行简短的描述。结果发现 TJA1040 基本上提供了与 PCA82C250/251 相同的功能。由于 TJA1050 和 TJA1040 的 CAN 信号都有良好的对称性，所以不需要一个专门的斜率控制模式。

（1）正常/高速模式

对于这里的所有驱动器而言，正常/高速模式都是相同的。它用于正常的 CAN 通信。从 TXD 输入的数字位流，被转换成相应的模拟总线信号。同时，驱动器监控总线，将模拟的总线信号转换成相应的数字位流，在 RXD 输出。

（2）待机模式

PCA82C250/251 和 TJA1040 提供了一个专用的待机模式。在这种模式中，电流消耗减到最低（如：TJA1040 最大<15μA，PCA82C250 最大<170μA）。专用的低功耗接收器确保了通过总线进行远程唤醒的功能。在待机模式中，TJA1040 和 PCA82C250/251 发送器完全禁止，而与 TXD 上的信号无关。这样 TJA1040 和 PCA82C250/251 提供了与“Babbling Idiot”节点一致的静音功能。TJA1040 和 PCA82C250/251 在这个模式下最大的区别是总线的偏压。PCA82C250/251 将总线偏压维持在 $V_{CC}/2$ 上，而 TJA1040 将总线拉到 GND。那么 TJA1040 在低功耗工作环境下的电流消耗会非常低。

（3）斜率控制模式

只有 PCA82C250/251 提供斜率控制模式。它通过在 Rs 引脚和 GND 电平之间连接电阻来调整斜率。由于 TJA1050 和 TJA1040 有优良的对称性，所以不需要斜率控制。它们都有一个固定的斜率，使用该斜率，TJA1050 和 TJA1040 的抗电磁干扰性比 PCA82C250/251 提高了 20dB。

（4）静音模式

TJA1050 提供一个专用的静音模式，这个模式中发送器完全禁止，这样就保证了没有信号能够从 TXD 发送到总线上。像 TJA1040 在待机模式一样，这个静音模式可以建立一个“Babbling Idiot”保护。静音模式中，接收器保持激活的状态，因此可以执行“只接收”功能。

图 4-11、图 4-12 和图 4-13 分别是实际应用中 PCA82C250/251、TJA1050 和 TJA1040 的不同应用电路：

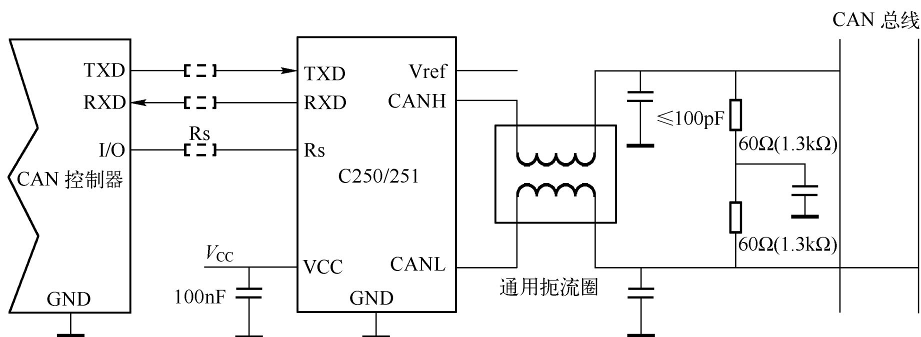


图 4-11　带有通用扼流圈的 PCA82C250/251 典型应用电路

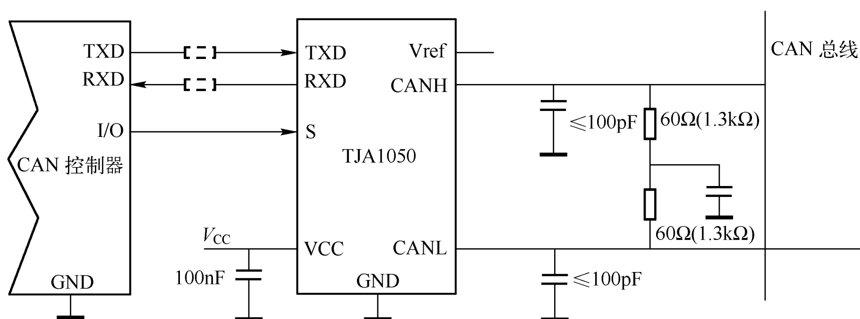


图 4-12　TJA1050 典型应用电路

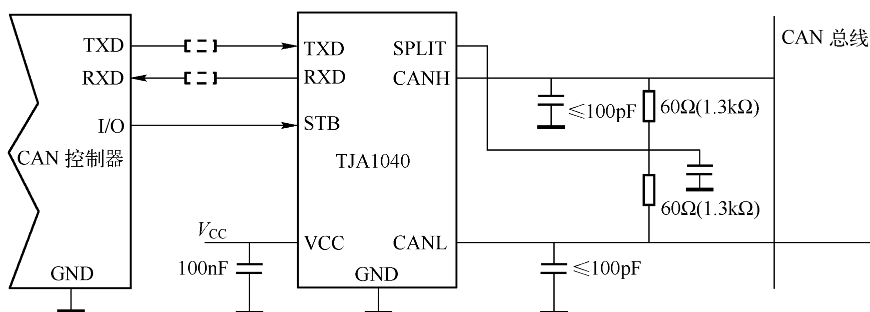


图 4-13　TJA1040 典型应用电路

4.6　本章小结

CAN 驱动器提供了 CAN 控制器与物理总线之间的接口以及对 CAN 总线的差动发送和接收功能，是影响系统网络性能的重要因素。实际应用中，根据不同的场合要选择不同类型的 CAN 总线驱动器。本章根据相关数据手册，针对具有代表性的 Philips 公司的 PCA82C250 和

PCA82C251 芯片，以及后继产品 TJA1040 和 TJA1050 分别进行了详细的对比介绍，以期对用户选型有些帮助。

思考题与习题

4.1　简要说明 CAN 驱动器在 CAN 总线网络中的作用。
4.2　比较 CAN 驱动器 82C250/52C251 的异同点。
4.3　比较 CAN 驱动器 TJA1040 和 TJA1050 的异同点。
4.4　请简单说明为什么在 CAN 总线的终端并上 120Ω的终端电阻。
4.5　设计 MCS-51 单片机、SJA1000 和 82C250 的接口电路，82C250 工作在高速模式。

第 5 章　具有 CAN 总线接口的微处理器及应用

以 C8051F50X 系列单片机、TMS320F2833X 系列 DSP 以及基于 ARM 内核的 STM32F107 微控制器为对象，介绍这些微处理器内部的 CAN 总线接口的硬件和软件设计知识。

本章要点

- C8051F50X 系列单片机的 CAN 接口及其应用；
- TMS320F2833X 系列 DSP 的 CAN 接口及其应用；
- 基于 ARM 内核的 STM32F107 微控制器 CAN 接口及其应用。

5.1　C8051F 系列单片机的 CAN 接口及其应用

5.1.1　C8051F50X 系列单片机介绍

C8051F50X 系列单片机是完全集成的混合信号片上的系统型单片机。它集成了数据采集或控制所需要的许多模拟和数字接口，代表了目前 8bit 单片机控制系统的发展方向。C8051F50X 单片机内部结构原理框图如图 5-1 所示。

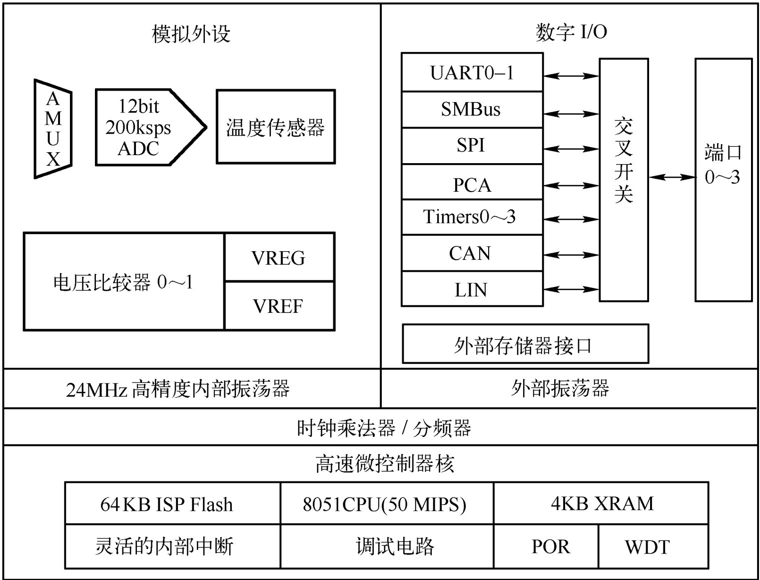


图 5-1　C8051F50X 单片机内部结构原理图

芯片上有 1 个 12bit 多通道 ADC，具有 40 个数字/模拟 I/O 引脚（C8051F500/1/4/5）或 25 个数字/模拟 I/O 引脚（C8051F502/3/6/7），一个 64KB（C8051F500/1/2/3）或者 32KB

（C8051F504/5/6/7）片上 flash 存储器，与 MCS-51 指令集完全兼容的高速 CIP-51 内核，它与 MCS-51 指令集完全兼容，可以使用标准 803x/805x 汇编器和编译器进行软件开发。CIP-51 内核具有标准 8051 的所有外设部件，并且还有硬件实现的 UART 串行接口和完全支持 CAN2.0A 和 CAN2.0B 协议的 CAN 控制器。

所采用的 CIP-51 微控制器内核采用流水线结构，与标准的 8051 结构相比指令执行速度有很大的提高。70%的指令的执行时间为 1 或 2 个系统时钟周期，只有 4 条指令的执行时间大于 4 个系统时钟周期。CIP-51 共有 111 条指令。CIP-51 工作在最大系统时钟频率 50MHz 时，其峰值性能达到 50MIPS。表 5−1 列出了指令条数与执行时所需的系统时钟周期数的关系。

表 5−1　C8051F50X CIP-51 内核指令条数与系统时钟周期数的关系

指令周期数	1	2	2/3	3	3/4	4	4/5	5	8
指令数	26	50	5	14	7	3	1	2	1

C8051F50X/51X 系列单片机采用 32 引脚或 48 引脚小封装。使得该系列产品非常适合于需要高性能和高集成度的紧凑型 PCB 尺寸的应用。该系列产品在汽车级温度范围内（−40～+125℃）的工作电压范围为 1.8～5.25V。其所有 I/O 端口和 RST 引脚耐 5V 输入，端口可配置为漏极开路输入或者推挽方式信号输出。

C8051F50X 系列 MCU 对 CIP-51 内核和外设有如下几项关键性的改进：

1）标准 8051 只有 7 个中断源，C8051F50X 系列单片机通过对内核中断系统的扩展，可向 CIP-51 提供 14 个中断源。允许大量的模拟和数字外设中断微控制器。

2）具有多达 8 个复位源：一个片内 VDD 监视器、一个看门狗定时器、一个时钟丢失检测器、一个由比较器 0 提供的电压检测器、一个软件强制复位、一个 flash 非法访问保护电路以及外部中断引脚 $\overline{\text{RST}}$ 复位。

3）内置一个独立运行的 24MHz 时钟发生器，其中 C8051F500/2/4/6 时钟精度可达±0.5%，C8051F501/3/5/7 时钟精度为±1.0%。内部时钟可根据需要进行分频或倍频编程调整，系统时钟可从 187.5kHz 调整到 48MHz。系统复位后默认内部时钟为系统时钟，如果需要，时钟源可以在运行时切换到外部振荡器，外部振荡器可以使用晶体、陶瓷谐振器、电容、RC 或外部时钟源来产生系统时钟。

4）内置了 Silicon Labs 两线开发接口（C2 口），该接口允许使用安装在最终产品上的单片机进行非侵入（不占用片上资源）、全速度、在线调试。该调试逻辑支持检查和修改存储器和寄存器、设置断点和观察点、单步、运行和停止命令。在使用 C2 进行调试时，所有模拟和数字外设都可全功能工作。

5.1.2　C8051F50X 内部 CAN 控制器介绍

C8051F50X 系列器件内集成了控制器局域网控制器，用 CAN 协议进行串行通信。Silicon Labs CAN 控制器符合 Bosch CAN 规范 2.0A（基本 CAN）和 2.0B（全功能 CAN），方便了在 CAN 网络上的通信。Silicon Labs CAN 是一个协议控制器，不提供物理层驱动器（即收发器）。其内部原理框图如图 5−2 所示。

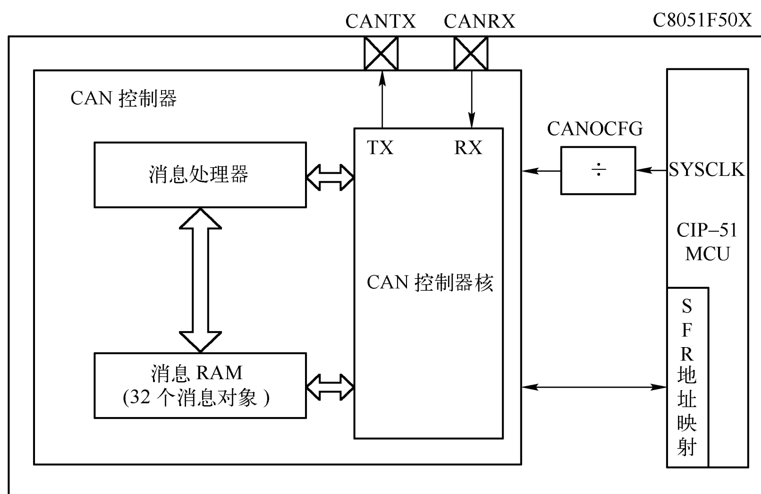


图 5-2　C8051F50X 系列 CAN 控制器原理框图

CAN 控制器包含一个 CAN 核、消息 RAM（独立于 CIP-51 的 RAM）、消息处理器和控制寄存器。Silicon Labs CAN 的工作位速率可达 1Mbit/s，实际速率可能受 CAN 总线上所选择的传输数据的物理层的限制。CAN 处理器有 32 个消息对象，可以被配置为发送或接收数据。输入数据、消息对象及其标识掩码存储在 CAN 消息 RAM 中。控制寄存器和消息处理器完成 CAN 控制器和 CIP-51 内核之间的数据和消息的传输，所有数据发送和接收过滤的协议处理全部由 CAN 控制器完成，不需 CIP-51 干预，使得用于 CAN 通信的 CPU 带宽最小。CIP-51 通过控制寄存器配置 CAN 控制器，读取接收到的数据和写入待发送的数据。

所有 C8051F50X 系列器件的 CAN 控制器都用特殊功能寄存器（SFR）来配置、监测和控制其外设。CAN 寄存器都位于特殊功能寄存器（SFR）的 0x0C 页，初始化 CAN 控制器的步骤如下：

① 将 SFRPAGE 寄存器设置为 0x0C；

② 将 CAN0CN 寄存器中的 INIT 和 CCE 位设置为 1；

③ 设置位定时寄存器和波特率预分频（BRP）扩展寄存器中的时序参数；

④ 初始化每个消息对象或者将其 MsgVal 位设置为无效；

⑤ 将 INIT 位清‘0’。

1. CAN 控制器时序

CAN 控制器的系统时钟（f_{sys}）来自 CIP-51 的系统时钟（SYSCLK）。由于 C8051F500/2/4/6 内部 24MHz 时钟精度可达±0.5%，C8051F501/3/5/7 时钟精度为±1.0%，按照 Bosch CAN 指导手册里的要求，使用不超过±1.58%误差的系统时钟，理论上可达到 125 kbit/s 的通信速率，因此在通信速率要求不高的场合，可直接采用内部时钟。C8051F50X 内部的 CAN 控制器不能在高于 24MHz 的频率下工作，因此当 CIP-51 系统时钟频率超过 24MHz 时，必须对 CAN 控制器中的 CAN0CFG 寄存器进行降频（分频）设置。

需要注意的是，由于 CAN 通信的高精度要求，一般需要使用外部振荡器（如石英晶体），才能达到 1M bit/s 的最高通信速率。系统时钟误差与通信速率的对应关系请参阅 Bosch CAN

用户指南（Bosch C_CAN User's Guide）。

2．CAN 寄存器访问

CAN 控制器中时钟参数配置寄存器中的 CAN 系统时钟分频系数的设置方法，会直接影响到 CAN 寄存器被访问的方式。如果时钟分频系数被置为 1，那么 CAN 的特殊功能寄存器在执行写入操作后可以立即被访问。如果分频系数不是 1，则 CAN 的特殊功能寄存器在被写入后就必须延迟一定系统周期后才能被读出。这种延迟可以通过执行类似 NOP 等不读取寄存器的指令来实现。这种访问限制主要体现在执行了写操作后立即进行读或者“读-写”操作指令上。所有受影响的指令有 ANL、ORL、MOV、XCH 和 XRL。

例如，当 CAN0CFG 中的分频系数置为 1 时，CAN0CN 可这样被访问：

```
MOV CAN0CN, #041              ;允许访问位定时寄存器
MOV R7, CAN0CN                ;将 CAN0CN 的值赋给 R7
```

但是当 CAN0CFG 中的分频系数置为 2 时，同样的指令中就需要额外增加一条 NOP 指令：

```
MOV CAN0CN, #041              ;允许访问位定时寄存器
NOP                           ;等待上一条写指令完成
MOV R7, CAN0CN                ;将 CAN0CN 的值赋给 R7
```

在上例中等待的周期数目依赖于分频系数的设置。分频系数为 N 时，就需要等待（N-1）个系统时钟周期才可以执行读操作。需要注意的是上述限制仅限于连续对同一个 CAN 寄存器进行先写后读操作，在对其他特殊功能寄存器进行访问时并不需要额外的延迟等待。

3．时序计算示例

本例说明如何配置 CAN 控制器的时序参数来满足 1M bit/s 的位速率。表 5-2 给出了进行计算所需要的与时序相关的系统参数。

表 5-2　CAN 控制器时序背景信息

参　　数	数　　值	说　　明
CIP-51 系统时钟（SYSCLK）/MHz	24	内部时钟振荡器
CAN 控制器系统时钟（f_{sys}）/MHz	24	CAN0CFG 除法系数设置为 1
CAN 时钟周期（t_{sys}）/ns	41.667	$1/f_{sys}$
CAN 时间量子（t_q）/ns	41.667	t_{sys} BRP①,②
CAN 总线长度/m	10	CAN 节点之间 5 ns/m 的信号延迟
传输延迟时间③/ns	400	2×（收发器环路延时+总线线路延时）

① CAN 时间量子（t_q）是 CAN 控制器能识别的最小时间单元。位时序参数通常用时间量子的整数倍给出。
② 波特率预分频器（BRP）被定义为 BRP 扩展寄存器中的值加 1。BRP 扩展寄存器的复位值为 0x0000；波特率预分频器的复位值为 0x0001。
③ 基于符合 ISO 11898 的收发器。CAN 对物理层没有规定。

CAN 网络上每个发送位有 4 个时段（SYNC-SEG, PROP-SEG，PHASE-SEG1 和 PHASE-SEG2），如图 5-3 所示。这些时段之和决定 CAN 的位时间（1/位速率）。在本例中，所期望的位速率为 1Mbit/s，因此位时间为 1000ns。

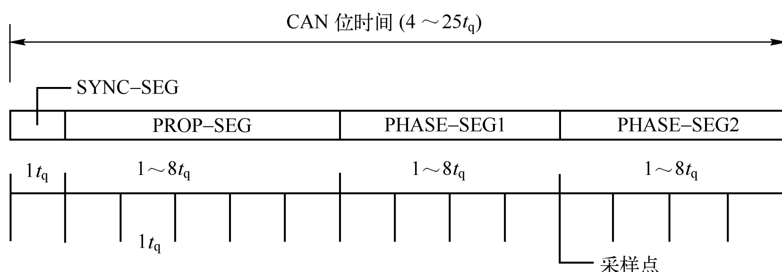


图 5-3　CAN 位时间的 4 个时段

为了使它们的和最接近所期望的位时间，需要对这 4 个位段的长度进行调整。由于每个时段必须是时间量子（t_q）的整数倍，所以可得到最接近的位时间为 24t_q（1 000.008ns），由此可得位速率为 0.999 992Mbit/s。SYNC−SEG 固定为 1t_q。PROP−SEG 必须大于或等于 400ns 的传输延迟时间，一般选 10t_q（416.67ns）。

位时间中剩余的时间量子数（13t_q）分配给 PHASE−SEG1 和 PHASE−SEG2，如图 5-3 所示。我们选 PHASE−SEG1=6t_q，PHASE−SEG2=7t_q。

$$\text{PHASE−SEG1} + \text{PHASE−SEG2} = \text{位时间} - (\text{SYNC−SEG} + \text{PROP−SEG}) \quad (5\text{−}1)$$

注 1：如果 PHASE−SEG1 + PHASE−SEG2 为偶数，则 PHASE−SEG2 = PHASE−SEG1。否则，PHASE−SEG2 = PHASE−SEG1 + 1。

注 2：PHASE−SEG2 至少应为 2t_q。

同步跳转宽度（SJW）这一时序参数由公式（5−2）定义。该参数用于确定写入到位定时寄存器中的数值和所期望的振荡器误差。由于我们使用石英晶体作为系统时钟源，所以不需要计算振荡误差容限。

$$\text{SJW} = \min(4, \text{PHASE−SEG1}) \quad (5\text{−}2)$$

写入到位定时寄存器中的值用公式（5−3）计算。BRP 扩展寄存器保持其复位值 0x0000 不变。

$$\begin{cases} \text{BRPE} = \text{BRP}-1 = \text{BRP 扩展寄存器} = \text{0x0000} \\ \text{SJWp} = \text{SJW} - 1 = \min(4, 6) - 1 = 3 \\ \text{TSEG1} = (\text{PROP−SEG} + \text{PHASE−SEG1}-1) = 10+6-1=15 \\ \text{TSEG2} = (\text{PHASE−SEG2} - 1) = 6 \\ \text{位时间寄存器} = \text{TSEG2} * \text{0x1000} + \text{TSEG1} * \text{0x0100} + \text{SJWp} * \text{0x0040} + \text{BRPE} = \text{0x6FC0} \end{cases} \quad (5\text{−}3)$$

5.1.3　C8051F50X 内部 CAN 寄存器介绍

CAN 寄存器分为如下 3 类：

1）CAN 控制器协议寄存器：CAN 控制、中断、错误控制、总线状态、测试方式。

2）消息对象接口寄存器：用于配置 32 个消息对象，向消息对象发送或从消息对象接收数据。

CIP-51 MCU 通过消息对象接口寄存器访问 CAN 消息 RAM。当向 IF1 或 IF2 命令请求寄存器写一个消息对象时，相关接口寄存器（IF1 或 IF2）的内容被传送到 CAN RAM 中的消息对象或消息对象被传送到接口寄存器。

3）消息处理器寄存器：这些只读寄存器用于向 CIP-51 MCU 提供有关消息对象的信息（MSGVLD 标志、发送请求标志、新数据标志）和中断标志（哪个消息对象引发了中断或状态中断条件）。

1．CAN 控制器协议寄存器

CAN 控制器协议寄存器用于配置 CAN 控制器，处理中断，监视总线状态，将 CAN 控制器置于测试模式。用 CIP-51 MCU 的 SFR 通过间接索引法访问 CAN 控制器协议寄存器，为了操作方便，某些寄存器可以用 CIP-51 的 SFR 直接访问。这些寄存器是：CAN 控制寄存器（CAN0CN）、CAN 状态寄存器（CAN0STA）、CAN 测试寄存器（CAN0TST）、错误计数寄存器、位定时寄存器及波特率预分频器（BRP）扩展寄存器。

2．消息对象接口寄存器

有两组消息对象接口寄存器，用于配置向 CAN 总线发送和从 CAN 总线接收数据的 32 个消息对象。消息对象可以被配置为发送或接收，并被分配消息标识，以便所有 CAN 节点进行接收过滤。消息对象保存在消息 RAM 中，用消息对象接口寄存器对其访问和配置。

3．消息处理器寄存器

消息处理器寄存器为只读寄存器。消息处理器寄存器提供中断、错误、发送/接收请求和新数据信息。

上述 3 类寄存器为标准的 Bosch CAN 寄存器，其与特殊功能寄存器空间的映射情况见表 5-3，可参考 Bosch CAN 用户手册中介绍的 CAN 控制协议寄存器的功能及用法。此外，C8051F50X 单片机 CAN 控制器还有一个特殊的寄存器 CAN0CFG，不属于标准的 Bosch CAN 寄存器，它用来配置 C8051F50X 单片机 CAN 控制器的时钟，其定义见表 5-4。所有 CAN 寄存器都位于特殊功能寄存器的 0x0C 页上。

表 5-3　标准的 Bosch CAN 寄存器及其复位值

CAN 地址	名　称	SFR 名称 (高字节)	SFR 地址	SFR 名称 (低字节)	SFR 地址	16bit SFR	复位值
0x00	CAN 控制寄存器	—	—	CAN0CN	0xC0	—	0x01
0x02	状态寄存器	—	—	CAN0STAT	0x94	—	0x00
0x04	错误寄存器①	CAN0ERRH	0x97	CAN0ERRL	0x96	CAN0ERR	0x0000
0x06	位定时寄存器②	CAN0BTH	0x9B	CAN0BTL	0x9A	CAN0BT	0x2301
0x08	中断寄存器①	CAN0IIDH	0x9D	CAN0IIDL	0x9C	CAN0IID	0x0000
0x0A	测试寄存器	—	—	CAN0TST	0x9E	—	0x00③,④
0x0C	BRP 扩展寄存器②	—	—	CAN0BRPE	0xA1	—	0x00
0x10	IF1 命令请求	CAN0IF1CRH	0xBF	CAN0IF1CRL	0xBE	CAN0IF1CR	0x0001
0x12	IF1 命令掩码	CAN0IF1CMH	0xC3	CAN0F1CML	0xC2	CAN0IF1CM	0x0000
0x14	IF1 掩码 1	CAN0IF1M1H	0xC5	CAN0IF1M1L	0xC4	CAN0IF1M1	0xFFFF
0x16	IF1 掩码 2	CAN0IF1M2H	0xC7	CAN0IF1M2L	0xC6	CAN0IF1M2	0xFFF
0x18	IF1 仲裁 1	CAN0IF1A1H	0xCB	CAN0IF1A1L	0xCA	CAN0IF1A1	0x0000
0x1A	IF1 仲裁 2	CAN0IF1A2H	0xCD	CAN0IF1A2L	0xCC	CAN0IF1A2	0x0000
0x1C	IF1 消息对象控制	CAN0IF1MCH	0xD3	CAN0IF1MCL	0xD2	CAN0IF1MC	0x0000
0x1E	IF1 数据 A1	CAN0IF1DA1H	0xD5	CAN0IF1DA1L	0xD4	CAN0IF1DA1	0x0000

（续）

CAN 地址	名　称	SFR 名称 (高字节)	SFR 地址	SFR 名称 (低字节)	SFR 地址	16bit SFR	复位值
0x20	IF1 数据 A2	CAN0IF1DA2H	0xD7	CAN0IF1DA2L	0xD6	CAN0IF1DA2	0x0000
0x22	IF1 数据 B1	CAN0IF1DB1H	0xDB	CAN0IF1DB1L	0xDA	CAN0IF1DB1	0x0000
0x24	IF1 数据 B2	CAN0IF1DB2H	0xDD	CAN0IF1DB2L	0xDC	CAN0IF1DB2	0x0000
0x40	IF2 命令请求	CAN0IF2CRH	0xDF	CAN0IF2CRL	0xDE	CAN0IF2CR	0x0001
0x42	IF2 命令掩码	CAN0IF1CMH	0xE3	CAN0IF2CML	0xE2	CAN0IF2CM	0x0000
0x44	IF2 掩码 1	CAN0IF2M1H	0xEB	CAN0IF2M1L	0xEA	CAN0IF2M1	0xFFFF
0x46	IF2 掩码 2	CAN0IF2M2H	0xED	CAN0IF2M2L	0xEC	CAN0IF2M2	0xFFFF
0x48	IF2 仲裁 1	CAN0IF2A1H	0xEF	CAN0IF2A1L	0xEE	CAN0IF2A1	0x0000
0x4A	IF2 仲裁 2	CAN0IF2A2H	0xF3	CAN0IF2A2L	0xF2	CAN0IF2A2	0x0000
0x4C	IF2 消息对象控制	CAN0IF2MCH	0xCF	CAN0IF2MCL	0xCE	CAN0IF2ML	0x0000
0x4E	IF2 数据 A1	CAN0IF2DA1H	0xF7	CAN0IF2DA1L	0xF6	CAN0IF2DA1	0x0000
0x50	IF2 数据 A2	CAN0IF2DA2H	0xFB	CAN0IF2DA2L	0xFA	CAN0IF2DA2	0x0000
0x52	IF2 数据 B1	CAN0IF2DB1H	0xFD	CAN0IF2DB1L	0xFC	CAN0IF2DB1	0x0000
0x54	IF2 数据 B2	CAN0IF2DB1H	0xFF	CAN0IF2DB1L	0xFE	CAN0IF2DB1	0x0000
0x80	发送请求 1①	CAN0TR1H	0xA3	CAN0TR1L	0xA2	CAN0TR1	0x0000
0x82	发送请求 2①	CAN0TR2H	0xA5	CAN0TR2L	0xA4	CAN0TR2	0x0000
0x90	新数据 1①	CAN0ND1H	0xAB	CAN0ND1L	0xAA	CAN0ND1	0x0000
0x92	新数据 2①	CAN0ND2H	0xAD	CAN0ND2L	0xAC	CAN0ND2	0x0000
0xA0	中断标志 1①	CAN0IP1H	0Xaf	CAN0IP1L	0xAE	CAN0IP1	0x0000
0xA2	中断标志 2①	CAN0IP2H	0xB3	CAN0IP2L	0xB2	CAN0IP2	0x0000
0xB0	消息有效 1①	CAN0MV1H	0XBB	CAN0MV1L	0xBA	CAN0MV1	0x0000
0xB2	消息有效 2①	CAN0MV1H	0XBD	CAN0MV1L	0xBC	CAN0MV1	0x0000

① 只读寄存器。
② 写使能被 CCE 控制。
③ CAN0TST 的复位值也可以为 r0000000b，其中 r 代表 CAN RX 引脚的值。
④ 写使能被 Test 控制。

表 5-4　CAN0CFG 的定义

位	名　称	功　能
7:2	未使用（可读）	Read=000000b;Write=闲置
1 : 0	SYSDIV[1 : 0]（可读/可写）	CAN 系统时钟除法器系数位 CAN 控制器时钟由 CIP-51 系统时钟分频得到，其最高频率不大于 25MHz 00：CAN 控制器时钟 ＝ 系统时钟/1 01：CAN 控制器时钟 ＝ 系统时钟/2 10：CAN 控制器时钟 ＝ 系统时钟/4 11：CAN 控制器时钟 ＝ 系统时钟/8
SFR Address=0x92；SFR Page=0x0C		

5.1.4　基于 C8051F500 的 CAN 硬件设计

基于 C8051F500 的 CAN 总线型拓扑结构如图 5-4 所示。CAN 节点的硬件电路如图 5-5

所示，使用 PCA82C250 作收发器。由于很多情况下，总线工作环境恶劣，为了进一步提高系统抗干扰能力，在 C8051F500 与 PCA82C250 之间增加高速光耦隔离器件 ADμM1201 构成隔离电路。在 CANH 和 CANL 上各自串联电阻 R2、R3 限流，再通过一组上下拉电阻 R4、R5，有效抑制反射波干扰，保持总线处于高阻态时，接收端收到的始终是“1”电平，这样可拉高信号的幅度，减少误码率。另外在 CANH 和 CANL 之间并联一对方向相反的瞬态二极管 VSl、VS2，可防雷击，以及防止总线上其他瞬变干扰。

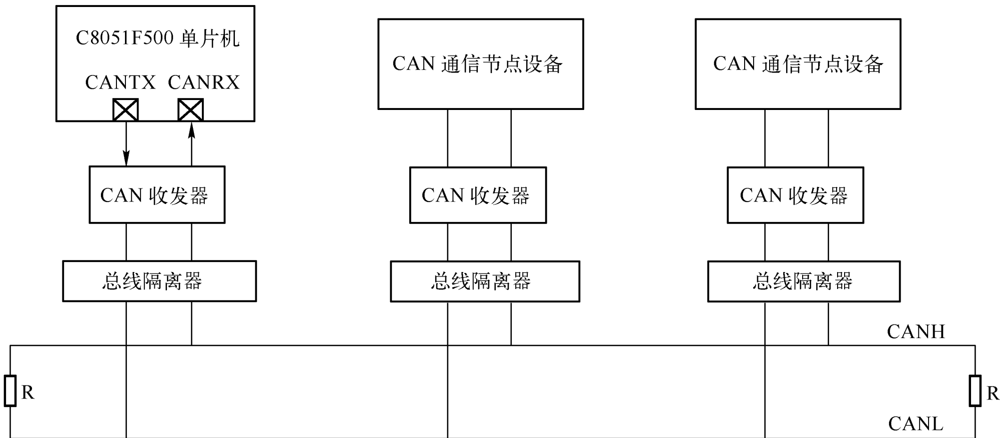


图 5-4　C8051F500 CAN 总线配置图

在 CANH 和 CANL 输出引脚间并联一个电阻，作为 CAN 总线的终端电阻，在本节点作 CAN 总线终端节点时，闭合跳线器 JPl，使终端电阻工作。终端电阻值 R6 等于传输电缆的特征阻抗，一般取值 120Ω，解决了远近端阻抗不匹配的影响，有助于降低信号反射。忽略掉它们，会使得数据通信的抗干扰性及可靠性大大降低。

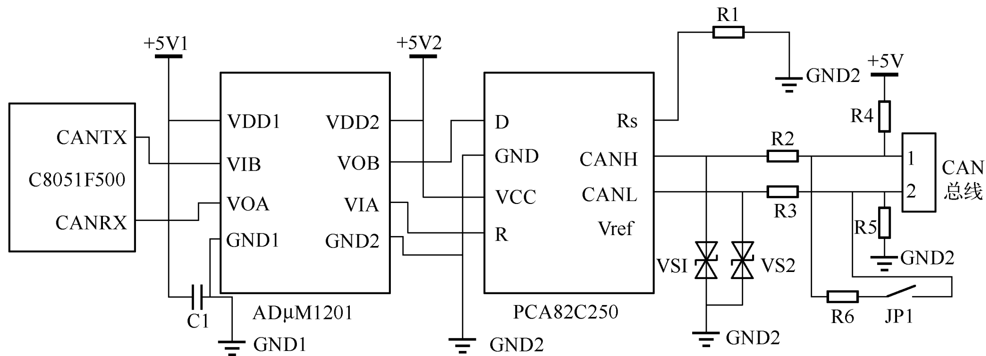


图 5-5　C8051F500 CAN 节点硬件电路图

5.1.5　基于 C8051F500 的 CAN 软件设计

具体的 CAN 总线通信网络的设计应包括应用层协议的制定，明确各个节点的功能，规定

每个数据位的含义以及其应做出的响应。图 5-5 所示的 CAN 节点电路所对应的软件设计主要包括 3 个主要部分：初始化程序，数据发送程序，数据接收程序。

1．初始化子程序

```
#include "C8051F500_defs.h"              //  调用 SFR 宏定义头文件
void CAN0_Init (void)
{
    U8 iter;
    U8 SFRPAGE_save = SFRPAGE;       //将 SFRPAGE 寄存器值备份
    SFRPAGE    = CAN0_PAGE;          //将 SFRPAGE 寄存器设置为 0x0C
    CAN0CN |= 0x01;                  //将初始化开始位置 1

    //初始化 CAN 消息对象
    //初始化 CAN 控制器
    CAN0CN |= 0x4E;                  //使能 CAN 时钟配置位，开启 CAN 模块中断
    CAN0BT = 0x6fc0;                 //系统时钟为 24MHz 时，配置 CAN 速率为 1Mbit/s

    //初始化命令掩码寄存器
    CAN0IF1CM = 0x00F0;              //配置为写操作
                                     //允许 CAN  屏蔽码，接收发送选择位，以及帧
                                     //格式写入 CAN 消息对象
    // CAN 掩码寄存器配置
    CAN0IF1M1 = 0x0000;              //  屏蔽未用于滤波的 16bit 识别码
    CAN0IF1M2 = 0x5FFC;              //  忽略扩展识别码
                                     //  使用方向位滤波
                                     //  使用 28～16bit 作为 CAN 识别码滤波
    //仲裁寄存器配置
    CAN0IF1A1 = 0x0000;              //配置 11bit 识别码，  不需要低 16bit 识别码

    //消息对象控制寄存器配置
    CAN0IF1MC = 0x0880 | MESSAGE_SIZE;        //使能接收中断
                                              //据宏定义头文件配置消息对象接收数据长度

    //对每个使用的消息对象进行配置
    for (iter = 0; iter < MESSAGE_OBJECTS; iter++)
    {   //仲裁寄存器配置
        CAN0IF1A2 = 0xA000 | (iter << 2);     //配置标识符，将消息对象配置位置位
        CAN0IF1CR = iter;                     //开始执行命令
    while (CAN0IF1CRH & 0x80)    {}           //等待配置完成
    }

    //对不使用的消息对象进行配置
    for (iter = MESSAGE_OBJECTS; iter < MESSAGE_OBJECTS; iter++)
    {
        CAN0IF1A2 = 0x0000;                   //消息对象配置位清零
        CAN0IF1CR = iter;                     //开始执行命令
```

```
        while (CAN0IF1CRH & 0x80) {}            //等待配置完成
    }
    CAN0CN &=  ~0x41;                           //返回正常模式并禁止访问位定时寄存器
    SFRPAGE = SFRPAGE_save;                     //将 SFRPAGE 寄存器值恢复至备份值
    CAN_TX_COMPLETE.U32 = 0x0000;               //初始化 CAN 控制器为无消息发送状态
}
```

2. CAN 发送子程序

```
void CAN0_TransferMO (U8 obj_num)
{
    // -----------初始化发送
    U8 SFRPAGE_save = SFRPAGE;
    SFRPAGE    = CAN0_PAGE;                     //将 SFRPAGE 寄存器设置为 0x0C
    CAN0IF1DA1L = CANTX_DATA[0];                //初始化发送数据寄存器
    CAN0IF1DA1H = CANTX_DATA[1];
    CAN0IF1DA2L = CANTX_DATA[2];
    CAN0IF1DA2H = CANTX_DATA[3];
    CAN0IF1DB1L = CANTX_DATA[4];
    CAN0IF1DB1H = CANTX_DATA[5];
    CAN0IF1DB2L = CANTX_DATA[6];
    CAN0IF1DB2H = CANTX_DATA[7];
    CAN0IF1CM = 0x0087;                         //置消息对象写操作标志
                                                //将数据写入消息对象
    CAN0IF1CR = obj_num;                        //开始执行命令
    while (CAN0IF1CRH & 0x80) {}                //等待发送完成
    SFRPAGE = SFRPAGE_save;
}
```

3. 接收中断服务程序

```
INTERRUPT (CAN0_ISR, INTERRUPT_CAN0)
{
    U8 iter;
    U8 carry;
    UU64 Rx_data;
    UU32 new_data;
    U8 status = CAN0STAT;                       // 读状态寄存器
    U8 Interrupt_ID = CAN0IID;                  // 读取引发本中断的中断标志
    new_data.U8[b0] = CAN0ND1L;                 // 接收数据
    new_data.U8[b1] = CAN0ND1H;
    new_data.U8[b2] = CAN0ND2L;
    new_data.U8[b3] = CAN0ND2H;
    carry = new_data.U8[b0] & 0x01;
    new_data.U32 = new_data.U32 >> 1;           //数据右移
    if (carry)
```

```
{
    new_data.U8[b3] = new_data.U8[b3] | 0x80;
}
CAN0IF1CM = 0x007F;                         //清中断标志
CAN0IF1CR = Interrupt_ID;                   //开始执行命令
while (CAN0IF1CRH & 0x80) {}                //等待命令执行完毕
if (status & RxOk)                          //判断是否成功接收数据
{
    //无论所有的 8B 的数据是否有效，都将其读出存储到 Rx_data 中
    Rx_data.U16[0] = CAN0IF1DA1;            //读取接收报文数据
    Rx_data.U16[1] = CAN0IF1DA2;
    Rx_data.U16[2] = CAN0IF1DB1;
    Rx_data.U16[3] = CAN0IF1DB2;
    if (Interrupt_ID = = 0x20)
    {
      Interrupt_ID = 0x00;         //由于 CIP-51 中消息对象 0 对应中断号为 0x20，因此将中
                                   //断标志相应设置为 0x00
    }
    for (iter = 0; iter < MESSAGE_SIZE; iter++)
     {
       if (Rx_data.U8[iter] != (Interrupt_ID + iter))
       {
           CAN_ERROR = 1;
       }
    }
    if (Interrupt_ID <= 15)
    {
         CAN_RX_COMPLETE.U32 |= (U16) (0x01 << Interrupt_ID);
    }
    else if (Interrupt_ID <= 0x1F)
    {
        CAN_RX_COMPLETE.U16[MSB] |= (U16) (0x01 << (Interrupt_ID - 16));
    }
}

//如果发生错误，设置错误状态变量
if (status & LEC)
{
    if ((status & LEC) != 0x07)    //识别错误类型，可相应增加处理措施
    {
        CAN_ERROR = 1;
    }
}

if (status & BOff)                 //判断总线是否处于关闭状态
```

```
    {
        CAN_ERROR = 1;
    }

    if (status & EWarn)                    //判断错误计数器是否处于警告状态
    {
        CAN_ERROR = 1;
    }
}
```

5.2 TMS320F28335 DSP 的 CAN 接口及其应用

5.2.1 TMS320F28335 介绍

TMS320C2000 系列是美国 TI 公司推出的最佳测控应用的定点 DSP 芯片，其主流产品分为 4 个系列：C20x、C24x、C27x 和 C28x。C20x 可用于通信设备、数字相机、嵌入式家电设备等；C24x 主要用于电机控制、工业自动化、电力转换系统等。近年来，TI 公司又推出了具有更高性能的改进型 C27x 和 C28x 系列芯片，进一步增强了芯片的接口能力和嵌入功能，从而拓宽了数字信号处理器的应用领域。

TMS320C2833x 系列是 TI 公司推出的 DSP 芯片，是目前国际市场上最先进、功能最强大的 32bit 浮点 DSP 芯片。它既具有数字信号处理能力，又具有强大的事件管理能力和嵌入式控制功能，特别适用于有大批量数据处理的测控场合，如工业自动化控制、电力电子技术应用、智能化仪器仪表及电机、伺服控制系统等。其硬件特征见表 5-5。

表 5-5　TMS320C2833x 系列 DSP 选型特征表

特　　征	F28335(150MHz)	F28334(150MHz)	F28332(150MHz)
指令周期（150MHz）/ ns	6.67	6.67	10
浮点单元	有	有	有
SRAM（16bit/字）KB	18	18	18
3.3V 片内 Flash（16bit/字）KB	256	128	64
一次可编程 ROM（16bit/字）KB	1	1	1
片内 Flash/SRAM 的密码	有	有	有
Boot ROM(8k×16)	有	有	有
16/32bit 外部中断接口	有	有	有
6 通道存储器直接读取	有	有	有
PWM 输出	Epwm1/2/3/4/5/6	Epwm1/2/3/4/5/6	Epwm1/2/3/4/5/6
高精度 PWM 输出	Epwm1A/2A/3A/4A/5A/6A	Epwm1A/2A/3A/4A/5A/6A	Epwm1A/2A/3A/4A
32bit 捕获或者辅助 PWM 输出	6	6	4
32bit 正交编码脉冲电路	2	2	2
看门狗定时器	有	有	有

（续）

特　　征		F28335(150MHz)	F28334(150MHz)	F28332(150MHz)
12bit 的 ADC	通道数	16	16	16
	吞吐率/MSPS	12.5	12.5	12.5
	转换时间/ns	80	80	80
32bit CPU 定时器		3	3	3
多通道缓冲串行接口(McBSP)/SPI		2	2	1
串行外部接口(SPI)		2	2	1
串行通信接口(SCI)		3	3	2
增强型 CAN 网络控制器(eCAN)		2	2	2
多功能复用 I/O 引脚		88	88	88
外部中断		8	8	8
封装	176 脚 PGF	有	有	有
	179 脚 ZHH	有	有	有
	176 脚 ZJZ	有	有	有
温度范围	A: −40℃～85℃	(PGF,ZHH,ZJZ)	(PGF,ZHH,ZJZ)	(PGF,ZHH,ZJZ)
	S: −40℃～125℃	(ZJZ)	(ZJZ)	(ZJZ)

TMS320F28335 是 C2833X 系列中比较主流的产品，它采用高性能静态 CMOS 技术，最高频率可达 150MHz（最小周期 6.67ns），下面列出了一些主要特性：

① 32bitCPU，IEEE-754 单精度 FPU，16×16 双 MAC；

② 6 通道 DMA 控制器；

③ 16/32bit 外部接口；

④ 12bit ADC，16 路外部输入，可选择内部或外部参考电压；

⑤ 片内 JTAG 调试和边界扫描，符合 IEEE1149.1-1990 边界扫描标准；

⑥ 具备仿真分析和断点功能，可通过硬件进行实时调试；

⑦ 内置 256k×16 Flash，34k×16 SARAM，Boot ROM (8k×16)；

⑧ 内置 3 个 32bit CPU 定时器；

⑨具备 128bit 安全密码/锁；

⑩ 最高 18bit PWM 输出，最高 6bit HRPWM 输出，最高 8 个 32bit/6 个 16bit 定时器；

⑪ 串行接口可支持 2 个 CAN 接口、3 个串行通信接口、2 个多通道缓冲串行口、1 个串行外围接口、1 个 I^2C 总线接口。

5.2.2　TMS320F28335 内部 eCAN 控制器介绍

在 TMS320F28335 中使用的增强型控制器区域网络（eCAN）模块与现行的 CAN2.0 标准兼容。它可使用已制定的协议在存在电子噪声的环境中与其他控制器进行串行通信。借助 32 个完全可配置的邮箱和时间标志特性，eCAN 模块提供了一种具有通用性和鲁棒性的串行通信接口。TMS320F28335 DSP 内部结构原理框图如图 5−6 所示。

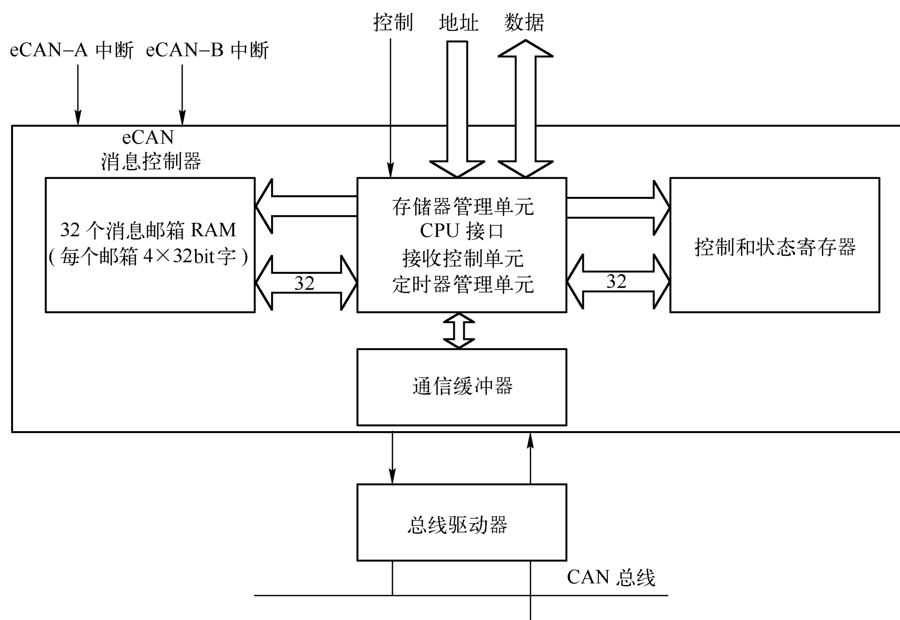


图 5-6　TMS320F28335 内部 eCAN 控制器框图

同时 TMS320F28335 也支持一种简化功能模式（SCC 模式），在这种模式中仅仅使用了 16 个邮箱（0～15），并省略了时间标志特性，减少了可用的接收屏蔽的数目。在默认状态下，选择该种模式。使用 SCB 位（CANMC）来选择 SCC 模式或者具有完整功能的 eCAN。

eCAN 是一种具有 32bit 内部结构的 CAN 控制器，eCAN 模块包括：

1）CAN 协议内核（CPK）。

2）消息控制器，主要由两部分组成，包括：

① 存储器管理单元，包括 CPU 接口和接收控制单元、定时器管理单元；

② 允许存储 32 个消息的邮箱 RAM。

在 CPK 接收到一个有效的消息后，消息控制器中的接收控制单元决定是否把接收的消息存储到邮箱 RAM 的 32 个邮箱中的某一个中。接收控制单元要检验状态、标识符和邮箱的屏蔽来决定适当的邮箱定位。接收的消息被存储到经过接收过滤后的第一个邮箱中。如果接收控制单元不能找到任何一个存储接收消息的邮箱，则消息被丢弃。

一个消息有 11 或 29bit 标识符，一个控制区域最多有 8B 的数据组成。当要发送一个消息时，消息控制器把这个消息发送到 CPK 的发送缓冲器中，以便在下一个总线空闲状态时开始发送消息。当多于一个消息要被发送时，具有最高优先级的消息将被消息控制器发送到 CPK 中。如果两个邮箱具有相同的优先级，则具有较大序号的邮箱将首先发送消息。

定时器管理单元包含一个时间标志计数器，所有的接收和发送的消息都要放置时间标志。在允许的时间间隔内（超时值），如果某个消息没有被接收或发送出去，则将产生一个中断。时间标志特性仅仅存在于 eCAN 模式中。

eCAN 模块有以下特性：

1）与 CAN2.0B 版协议完全兼容。

2）支持高达 1Mbit/s 的传输速率。

3）具有 32 个邮箱，每个邮箱都具有以下特性：

① 可配置为接收或发送；

② 可用标准的或扩展的标识符进行配置；

③ 具有可编程的接收过滤屏蔽；

④ 支持数据帧和远程帧；

⑤ 支持 0～8B 的数据；

⑥ 在接收和发送的消息中使用一个 32bit 的时间标志；

⑦ 阻止旧消息被新消息覆盖的保护措施；

⑧ 允许对发送消息优先级的动态编程；

⑨ 使用一种具有两个中断级别的可编程的中断方案；

⑩ 对发送和接收的超时现象使用一种可编程的中断操作。

4）低功耗模式。

5）总线活动的可编程唤醒。

6）远端请求消息的自动应答。

7）某帧在缺少仲裁和发生错误情况下的自动重传。

8）有一个特殊消息同步的 32bit 时间标志计数器。

9）自检模式。

10）以一种回环的方式接收自己的消息。提供了一种虚拟的响应信息，从而不必要求另一个节点提供响应位。

11）支持用于通信的 4 种不同类型的帧：

① 数据帧；

② 远程帧；

③ 错误帧；

④ 超载帧。

5.2.3 TMS320F28335 内部 eCAN 寄存器介绍

eCAN 模块在 TMS320F28335 存储器中有两种不同的地址段映射方式。第一个地址段被用于访问控制寄存器、状态寄存器、接收屏蔽、时间标志和消息对象的超时寄存器。对控制和状态寄存器的访问被限制为 32bit 宽度的存取。本地的接收屏蔽、时间标志寄存器和超时寄存器可以存取 8bit、16bit 和 32bit 宽度的数据。第二个地址段被用于访问邮箱，这个存储器区域也可以存取 8bit、16bit 和 32bit 宽度的数据。这两个存储器模块都是用 512B 的地址空间。如图 5-7 所示的 eCAN-A 的存储器映射。eCAN-B 具有与 eCAN-A 相同的存储器映射结构，只是地址不同。

（1）邮箱使能寄存器(CANME)

邮箱使能寄存器位说明见表 5-6。

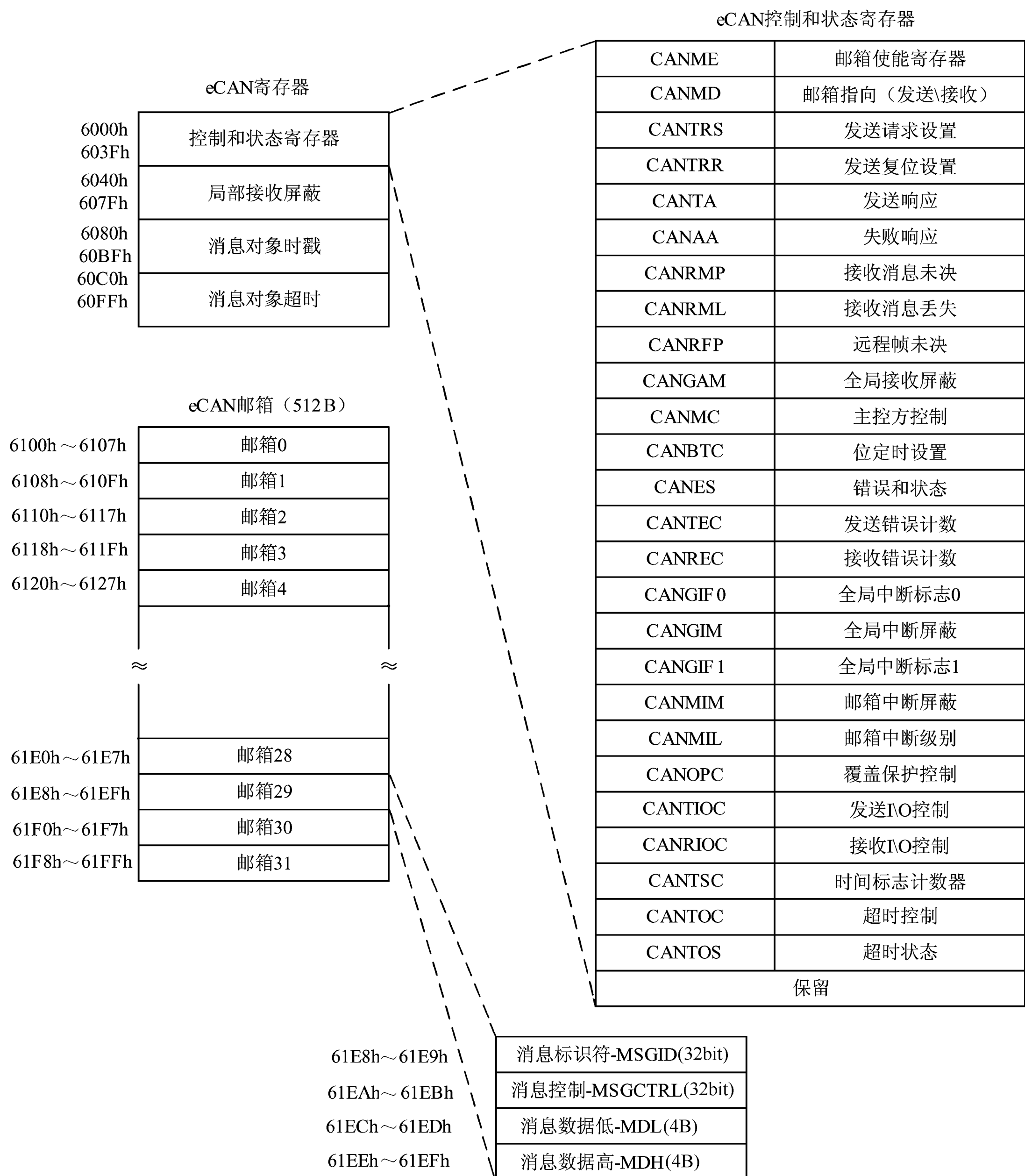


图 5-7　eCAN-A 存储器映射关系表

表 5-6　邮箱使能寄存器位说明

位	名　称	位操作权限	复　位　值	说　明
31～0	ME[31 : 0]	可读/可写	0	邮箱使能位。上电后， CANME 中所有比特位都被清 0。非使能邮箱可被 CPU 作为附加的存储器 1：对应的邮箱对 CAN 模块使能 0：对应的邮箱 RAM 区域对 eCAN 禁用

（2）邮箱指向寄存器（CANMD）

邮箱指向寄存器位说明见表 5-7。

表 5-7　邮箱指向寄存器位说明

位	名　称	位操作权限	复 位 值	说　明
31～0	MD[31∶0]	可读/可写	0	邮箱方向寄存器： 1：对应的邮箱被定义为接收邮箱 0：对应的邮箱被定义为发送邮箱

（3）发送请求置位寄存器（CANTRS）

当邮箱 n 准备发送数据时，CPU 设置 TRS[n]位为 1 以启动发送。发送请求置位寄存器位说明见表 5-8。

表 5-8　发送请求置位寄存器位说明

位	名　称	位操作权限	复 位 值	说　明
31～0	TRS[31∶0]	可读/设置	0	发送请求置位： 1：置位该位会发送在相应邮箱中的消息。可以同时置位几个位而使多个消息轮流发送 0：无操作

（4）发送请求复位寄存器（TRR）

这些位仅能通过 CPU 置位和通过内部逻辑复位。发送请求复位寄存器位说明见表 5-9。

表 5-9　发送请求复位寄存器位说明

位	名　称	位操作权限	复 位 值	说　明
31～0	TRR[31∶0]	可读/设置	0	发送请求复位位： 1：置位 TRRn 会取消一个发送请求 0：无操作

（5）发送响应寄存器（CANTA）

如果邮箱 n 中的消息被成功发送，则 TA[n]被置位。发送响应寄存器位说明见表 5-10。

表 5-10　发送响应寄存器位说明

位	名　称	位操作权限	复 位 值	说　明
31～0	TA[31∶0]	可读/清除	0	发送响应位： 1：如果邮箱 n 中的消息被成功发送，则该寄存器的位 n 被置位 0：消息没有被发送

（6）失败响应寄存器（CANAA）

如果邮箱 n 中的消息发送失败，则 AA[n]位和 AAIF（GIF14）位将被置位，同时也会产生中断（如果中断已设为使能）。失败响应寄存器位说明见表 5-11。

表 5-11　失败响应寄存器位说明

位	名　称	位操作权限	复 位 值	说　明
31～0	AA[31∶0]	可读/清除	0	失败响应位： 1：如果邮箱 n 中的消息发送失败，则该寄存器的位 n 被置位 0：发送没有失败

（7）接收消息未决寄存器（CANRMP）

如果邮箱 n 中包含一个接收到的消息，则该寄存器的 RMP[n]位被置位。接收消息未决寄存器位说明见表 5-12。

表 5-12　接收消息未决寄存器位说明

位	名　称	位操作权限	复 位 值	明
31～0	RMP[31∶0]	可读/清除	0	接收消息未决位： 1：如果邮箱 n 中包含一个已接收的消息，则该寄存器的位 RMP[n]被置位 0：邮箱不包含消息

（8）接收消息丢失寄存器（CANRML）

如果邮箱 n 中原有的消息被一个新的消息所覆盖，则该寄存器的 RML[n]位被置位。接收消息丢失寄存器位说明见表 5-13。

表 5-13　接收消息丢失寄存器位说明

位	名　称	位操作权限	复 位 值	说　明
31～0	RML[31∶0]	可读/清除	0	接收消息丢失位： 1：某邮箱中的一个旧的未被及时读取的消息已被一个新的消息所覆盖 0：没有丢失消息

（9）远程帧未决寄存器（CANRFP）

无论何时只要 CAN 模块接收到一个远程帧请求，则远程帧未决寄存器中的对应位 RFP[n]就将被置位。远程帧未决寄存器位说明见表 5-14。

表 5-14　远程帧未决寄存器位说明

位	名　称	位操作权限	复 位 值	说　明
31～0	RFP[31∶0]	可读/清除	0	远程帧未决寄存器： 1：模块接收到一个远程帧 0：没有接收到远程帧。该寄存器被 CPU 清 0

（10）全局接收屏蔽寄存器（CANGAM）

全局接收屏蔽用于 eCAN 的 SCC 模式。全局接收屏蔽用于 SCC 模式的 6～15 号邮箱。全局接收屏蔽寄存器位说明见表 5-15。

表 5-15　全局接收屏蔽寄存器位说明

位	名　称	位操作权限	复 位 值	说　明
31	AMI	在任何时间可读/仅在初始化模式期间可写	0	接收屏蔽标识符扩展位： 1：可以接收标准和扩展帧 0：存储在邮箱中的标识符扩展位决定着哪一个消息要被接收
30～29	Reserved	可读	0	读操作为不确定值，写操作无效
28～0	GAM[28∶0]	在任何时间可读/仅在初始化模式期间可写	0	全局接收屏蔽位

（11）主控制寄存器（CANMC）

CANMC 用于控制 CAN 模块的设置。主控制器寄存器位说明见表 5-16。

表 5-16　主控制器寄存器位说明

位	名称	位操作权限	复位值	说明
31～17	Reserved	可读	0	读操作为不确定值，写操作无效
16	SUSP	可读/可写	0	暂停模式位
15	MBCC	可读/仅在 EALLOW 模式可写	0	邮箱时间标志计数器清 0 位
14	TCC	仅在 EALLOW 模式可设置	不确定	时间标志计数器 MSB 清 0 位
13	SCB	可读/仅在 EALLOW 模式下可写	0	SCC 兼容位
12	CCR		1	变换配置请求位
11	PDR		0	功率下降模式请求位
10	DBO		0	数据字节顺序
9	WUBA		0	总线唤醒位
8	CDR		0	改变数据区域请求位
7	ABO		0	自动开启总线
6	STM		0	自检模式
5	SRES	可读/仅在 EALLOW 模式可设置	0	这一位只能写入，读出的数据始终为 0
4～0	MBNR	可读/可写	0	邮箱号

（12）位定时配置寄存器（CANBTC）

这个寄存器可以对 CAN 节点进行配置。位定时配置寄存器位说明见表 5-17。

表 5-17　位定时配置寄存器位说明

位	名　　称	位操作权限	复　位　值	说　　明
31～24	Reserved	可读	不确定	读操作为不确定值，写操作无效
23～16	BRPreg	可读/仅在 EALLOW 模式下的初始化过程中可写	0	波特率预定标器
15～10	Reserved	可读	0	保留位
9～8	SJWreg	可读/仅在 EALLOW 模式下的初始化过程中可写	0	同步跳跃宽度
7	SAM		0	设置 CAN 模块的采样数目
6～3	TSEG1reg		0	时间段 1
2～0	TSEG2reg		0	时间段 2

（13）错误和状态寄存器（CANES）

CAN 模块的状态由错误和状态寄存器和错误计数寄存器来描述。

（14）错误计数寄存器（CANTEC）

CAN 模块包含两个错误计数器：接收错误计数器和发送错误计数器。通过 CPU 接口可以读取它们的值。

（15）中断寄存器

① 全局中断标志寄存器（CANGIF0/CANGIF1）；

② 全局中断屏蔽寄存器（CANGIM）；

③ 邮箱中断屏蔽寄存器（CANMIM）。

（16）覆盖保护控制寄存器（CANOPC）

如果邮箱 n 出现溢出，新消息是否存储取决于寄存器 CANOPC 的设置。

（17）eCANI/O 控制寄存器（CANTIOC）

CANTX 和 CANRX 引脚在经过配置后，可用于 CAN 模块。可使用 CANTIOC 和 CANRIOC 进行配置。

（18）定时器管理单元

在发送或接收消息时，利用 eCAN 的某些功能可以监视操作时间。在 eCAN 中具有一个单独的状态机来处理“控制时间”的功能。所以，“控制时间”功能可能被其他正在进行的操作所延迟。

① 时间标志功能：时间标志计数器寄存器（CANTSC）、邮箱时间标志寄存器（MOTS）；

② 超时功能：邮箱超时寄存器（MOTO）、超时控制寄存器（CANTOC）和超时状态寄存器（CANTOS）；

③ MTOF0/1 位在用户应用中的作用。

（19）邮箱设计寄存器

每一个邮箱有包含 4 个 32bit 的寄存器：消息标识寄存器（MSGID）、消息控制寄存器（MSGCTRL）、消息数据寄存器（MDL 和 MDH）。

（20）接收过滤器

输入消息的标识符首先与邮箱的标识符（存在于邮箱中）进行比较。于是，可以利用适当的接收屏蔽方法来屏蔽掉那些不需进行比较的标识符位。

对于 SCC 兼容模式，全局接收屏蔽（GAM）寄存器被用于 6～15 号邮箱。一个输入的消息将被存储到具有匹配标识符而且具有最大编号的邮箱中。如果 6～15 号邮箱中没有匹配的标识符，则输入的消息会再与存储在 5～3 号邮箱中的标识符进行比较，而后是 2～0 号邮箱。

在 eCAN 的 32 个邮箱中，每一个都有自己的局部接收屏蔽位 LAM(0)～LAM(31)。在 eCAN 模式中没有全局接收屏蔽位。

5.2.4 基于 TMS320F28335 的 CAN 硬件设计

基于 TMS320F28335 的 CAN 节点硬件电路如图 5-8 所示。电路使用 SN65HVD230 作为收发器。

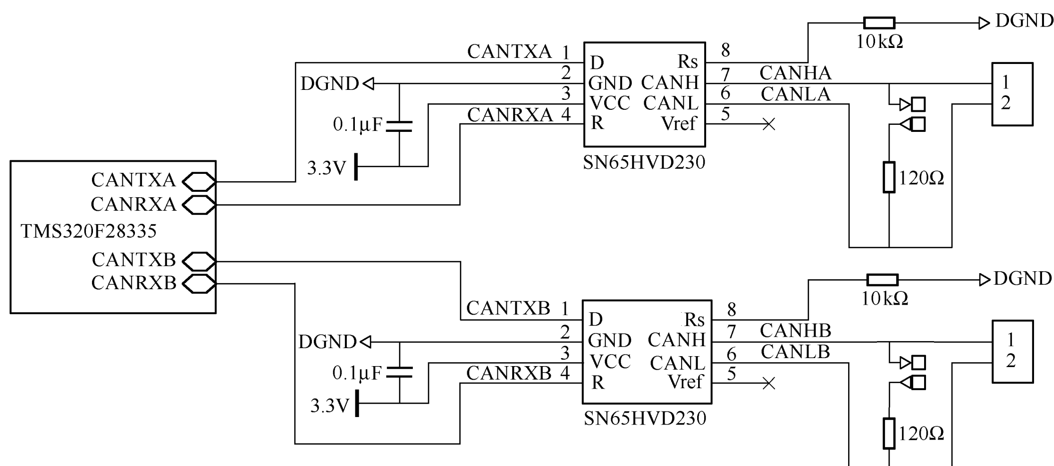


图 5-8　TMS320F28335 CAN 节点硬件电路图

5.2.5 基于 TMS320F28335 的 CAN 软件设计

1. 初始化，以 eCAN-A 模块的初始化和报文读取程序作为示例。

```
void InitECana(void)                          // 初始化 CAN-A 模块
{
    struct ECAN_REGS ECanaShadow;             //不能位操作
    EALLOW;                                   // 去掉寄存器保护
//通过 eCAN 的寄存器配置芯片的引脚
    ECanaShadow.CANTIOC.all = ECanaRegs.CANTIOC.all;
    ECanaShadow.CANTIOC.bit.TXFUNC = 1;
    ECanaRegs.CANTIOC.all = ECanaShadow.CANTIOC.all;
    ECanaShadow.CANRIOC.all = ECanaRegs.CANRIOC.all;
    ECanaShadow.CANRIOC.bit.RXFUNC = 1;
    ECanaRegs.CANRIOC.all = ECanaShadow.CANRIOC.all;
//配置 eCAN 为 HECC 模式（允许使用 0～31 号邮箱）
    ECanaShadow.CANMC.all = ECanaRegs.CANMC.all;
    ECanaShadow.CANMC.bit.SCB = 1;
    ECanaShadow.CANMC.bit.STM = 0;
    ECanaRegs.CANMC.all = ECanaShadow.CANMC.all;
    EDIS;                                     // 恢复寄存器保护功能
//将消息标识符寄存器初始化为 0
    ECanaRegs.CANME.all = 0;
//配置发送邮箱
    ECanaMboxes.MBOX0.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX1.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX2.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX3.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX4.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX5.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX6.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX7.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX8.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX9.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX10.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX11.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX12.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX13.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX14.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX15.MSGID.all = 0x20000000;
//配置接收邮箱
    ECanaMboxes.MBOX16.MSGID.all=0x20000000;
    ECanaMboxes.MBOX17.MSGID.all=0x20000000;
    ECanaMboxes.MBOX18.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX19.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX20.MSGID.all = 0x20000000;
```

```
    ECanaMboxes.MBOX21.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX22.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX23.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX24.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX25.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX26.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX27.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX28.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX29.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX30.MSGID.all = 0x20000000;
    ECanaMboxes.MBOX31.MSGID.all = 0x20000000;
//配置邮箱发送数据长度
    ECanaMboxes.MBOX0.MSGCTRL.bit.DLC = 8;
    ECanaMboxes.MBOX1.MSGCTRL.bit.DLC = 8;
    ECanaMboxes.MBOX2.MSGCTRL.bit.DLC = 8;
    ECanaMboxes.MBOX3.MSGCTRL.bit.DLC = 8;
    ECanaMboxes.MBOX4.MSGCTRL.bit.DLC = 8;
    ECanaMboxes.MBOX5.MSGCTRL.bit.DLC = 8;
    ECanaMboxes.MBOX6.MSGCTRL.bit.DLC = 8;
    ECanaMboxes.MBOX7.MSGCTRL.bit.DLC = 8;
    ECanaMboxes.MBOX8.MSGCTRL.bit.DLC = 8;
    ECanaMboxes.MBOX9.MSGCTRL.bit.DLC = 8;
    ECanaMboxes.MBOX10.MSGCTRL.bit.DLC = 8;
    ECanaMboxes.MBOX11.MSGCTRL.bit.DLC = 8;
    ECanaMboxes.MBOX12.MSGCTRL.bit.DLC = 8;
    ECanaMboxes.MBOX13.MSGCTRL.bit.DLC = 8;
    ECanaMboxes.MBOX14.MSGCTRL.bit.DLC = 8;
    ECanaMboxes.MBOX15.MSGCTRL.bit.DLC = 8;
//远程发送请求位清零
    ECanaMboxes.MBOX0.MSGCTRL.bit.RTR = 0;
    ECanaMboxes.MBOX1.MSGCTRL.bit.RTR = 0;
    ECanaMboxes.MBOX2.MSGCTRL.bit.RTR = 0;
    ECanaMboxes.MBOX3.MSGCTRL.bit.RTR = 0;
    ECanaMboxes.MBOX4.MSGCTRL.bit.RTR = 0;
    ECanaMboxes.MBOX5.MSGCTRL.bit.RTR = 0;
    ECanaMboxes.MBOX6.MSGCTRL.bit.RTR = 0;
    ECanaMboxes.MBOX7.MSGCTRL.bit.RTR = 0;
    ECanaMboxes.MBOX8.MSGCTRL.bit.RTR = 0;
    ECanaMboxes.MBOX9.MSGCTRL.bit.RTR = 0;
    ECanaMboxes.MBOX10.MSGCTRL.bit.RTR = 0;
    ECanaMboxes.MBOX11.MSGCTRL.bit.RTR = 0;
    ECanaMboxes.MBOX12.MSGCTRL.bit.RTR = 0;
    ECanaMboxes.MBOX13.MSGCTRL.bit.RTR = 0;
    ECanaMboxes.MBOX14.MSGCTRL.bit.RTR = 0;
    ECanaMboxes.MBOX15.MSGCTRL.bit.RTR = 0;
    ECanaRegs.CANMD.all = 0xFFFF0000;
```

```
    ECanaRegs.CANME.all = 0xFFFFFFFF;
//TAn, RMPn, GIFn 位清零
    ECanaRegs.CANTA.all  = 0xFFFFFFFF;              //清除发送相应寄存器
    ECanaRegs.CANRMP.all = 0xFFFFFFFF;              //清除消息寄存器
    ECanaRegs.CANGIF0.all = 0xFFFFFFFF;             //清除全局中断标志寄存器
    ECanaRegs.CANGIF1.all = 0xFFFFFFFF;
    ECanaRegs.CANOPC.all=0xFFFF0000;
//配置 eCANA 时钟参数
    EALLOW;
    ECanaRegs.CANMIM.all= 0xFFFFFFFF;
    ECanaShadow.CANMC.all = ECanaRegs.CANMC.all;
    ECanaShadow.CANMC.bit.CCR = 1 ;                 // 将变换配置请求位置 1
    ECanaRegs.CANMC.all = ECanaShadow.CANMC.all;
    EDIS;
    ECanaShadow.CANES.all = ECanaRegs.CANES.all;
    do      {
    ECanaShadow.CANES.all = ECanaRegs.CANES.all;
    } while(ECanaShadow.CANES.bit.CCE != 1 );       // 等待配置完成
    EALLOW;
    ECanaShadow.CANBTC.all = ECanaRegs.CANBTC.all;
    #if (CPU_FRQ_150MHz)
//系统时钟为 150MHz 时，配置默认的位速率为 1 Mbit/s
    ECanaShadow.CANBTC.bit.BRPREG = 9;
    ECanaShadow.CANBTC.bit.TSEG2REG = 2;
    ECanaShadow.CANBTC.bit.TSEG1REG = 10;
    #endif
    #if (CPU_FRQ_100MHz)
//系统时钟为 100MHz 时，配置默认的位速率为 1 Mbit/s
    ECanaShadow.CANBTC.bit.BRPREG = 9;
    ECanaShadow.CANBTC.bit.TSEG2REG = 1;
    ECanaShadow.CANBTC.bit.TSEG1REG = 6;
    #endif
    ECanaShadow.CANBTC.bit.SAM = 1;
    ECanaRegs.CANBTC.all = ECanaShadow.CANBTC.all;
    ECanaShadow.CANMC.all = ECanaRegs.CANMC.all;
    ECanaShadow.CANMC.bit.CCR = 0 ;             // 将变换配置请求位置 1
    ECanaShadow.CANMC.bit.PDR = 0 ;             // 选择普通操作模式
    ECanaShadow.CANMC.bit.DBO = 0 ;             // 接收或发送数据的字节顺序配置位
    ECanaShadow.CANMC.bit.WUBA = 0 ;            // 总线唤醒位清零
    ECanaShadow.CANMC.bit.CDR = 0 ;             // 改变数据区域请求位清零
    ECanaShadow.CANMC.bit.ABO = 0 ;             // 关闭自动开启总线
    ECanaShadow.CANMC.bit.STM = 0 ;             // 关闭自检模式
    ECanaShadow.CANMC.bit.SRES = 0 ;            // 软件复位位清零
    ECanaShadow.CANMC.bit.MBNR = 0;
    ECanaRegs.CANMC.all = ECanaShadow.CANMC.all;
    ECanaShadow.CANES.all = ECanaRegs.CANES.all;
```

```
        EDIS;
        do
        {
            ECanaShadow.CANES.all = ECanaRegs.CANES.all;
        } while(ECanaShadow.CANES.bit.CCE != 0 );          // 等待改变配置使能清 0
         EALLOW;
         ECanaRegs.CANGIM.bit.I0EN = 1;
         EDIS;                                              // 禁止所有邮箱
}
```

2. 发送程序

```
void Mailbox_Send(Uint16 num,Uint32 message[2])
//仅给出 0 号邮箱的发送程序，使用 1～15 号邮箱进行发送可采用类似的代码
{
        switch(num)
        {
                case 0:
                        ECanaMboxes.MBOX0.MDH.all= message[0];
                        ECanaMboxes.MBOX0.MDL.all= message[1];
                        ECanaRegs.CANTRS.bit.TRS0 = 1;
                        ECanaRegs.CANTA.bit.TA0 = 1;
                        break;
                case 1:ECanaRegs.CANTRS.bit.TRS1= 1;break;
                case 2:ECanaRegs.CANTRS.bit.TRS2 = 1;break;
                case 3:ECanaRegs.CANTRS.bit.TRS3 = 1;break;
                case 4:ECanaRegs.CANTRS.bit.TRS4 = 1;break;
                case 5:ECanaRegs.CANTRS.bit.TRS5 = 1;break;
                case 6:ECanaRegs.CANTRS.bit.TRS6 = 1;break;
                case 7:ECanaRegs.CANTRS.bit.TRS7 = 1;break;
                case 8:ECanaRegs.CANTRS.bit.TRS8 = 1;break;
                case 9:ECanaRegs.CANTRS.bit.TRS9 = 1;break;
                case 10:ECanaRegs.CANTRS.bit.TRS10 = 1;break;
                case 11:ECanaRegs.CANTRS.bit.TRS11 = 1;break;
                case 12:ECanaRegs.CANTRS.bit.TRS12 = 1;break;
                case 13:ECanaRegs.CANTRS.bit.TRS13 = 1;break;
                case 14:ECanaRegs.CANTRS.bit.TRS14 = 1;break;
                case 15:ECanaRegs.CANTRS.bit.TRS15 = 1;break;
                case 16:break;
                default:break;
        }
        return;
}
```

3. 接收程序

```
Uint32    TestMbox1 = 0;
Uint32    TestMbox2 = 0;
```

```
Uint32    TestMbox3 = 0;
void mailbox_read(int16 MBXnbr)
{
    volatile struct MBOX *Mailbox;              //定义指针变量指向要读取的邮箱地址
    Mailbox = &ECanaMboxes.MBOX0 + MBXnbr;
    TestMbox1 = Mailbox->MDL.all;               //读消息数据寄存存器（MDL,MDH）
    TestMbox2 = Mailbox->MDH.all;
    TestMbox3 = Mailbox->MSGID.all;             //读消息标识寄存存器
}
```

5.3 基于 ARM® Cortex™-M3 内核的 STM32F107 微控制器 CAN 接口及其应用

5.3.1 STM32F107 芯片介绍

Cortex™-M3 内核是 ARM 公司为了满足集高性能、低功耗、实时应用、具有竞争性价格于一体的嵌入式领域的要求而推出的新型内核。作为 ARM7 的后继者，Cortex™-M3 大幅度改革了设计架构，简化了编程和调试的复杂度，增强了处理能力，并引入了很多专门满足单片机应用程序需求的崭新技术。

STM32 系列是意法半导体公司基于 Cortex™-M3 内核推出的新型 32bit 闪存微处理器，具有优异的实时性能、杰出的功耗控制以及出众且创新的外设，可以实现产品最大程度的集成整合，易于开发，可使产品快速进入市场。

STM32F107 是 STM32 系列中的互联性产品，其最高时钟频率可达 72MHz，并提供 256KB 的 Flash 存储空间和 64KB 的 SRAM 空间。该处理器最大的特色是提供最多 14 个丰富的通信外设，包括 2 个 I2C 总线控制器、5 个 UARTS 控制器、3 个 SPI 控制器（最高速度 18Mbit/s，其中两个可以用作 I2S 接口）、2 个 CAN 总线控制器、一个 USB2.0 全速 OTG 控制器和一个 10Mbit/s、100Mbit/s 以太网控制器，可以方便地应用于需要多种通信方式的嵌入式设备。

5.3.2 STM32F107 的 CAN 控制器概述

STM32F107 中共有两个 CAN 控制器，分别为主控制器 CAN1 和从控制器 CAN2，均支持 CAN2.0A 与 2.0B 协议，最高通信波特率可达 1Mbit/s。两个控制器共享 512B 的 SRAM 存储空间，供发送和接收时的缓冲区以及标识符过滤槽使用，其中主控制器 CAN1 除了进行正常的 CAN 通信之外，还要负责管理这 512B 的存储空间，从控制器 CAN2 只负责 CAN 通信，而无权操作 SRAM 存储空间，如图 5-9 所示。

在作为发送方时，STM32F107 中的每个 CAN 控制器均提供 3 个发送邮箱，用户软件通过将数据写入发送邮箱的方式来实现 CAN 数据帧的发送，这些邮箱的发送优先级可以配置，控制器会根据这些邮箱的优先级对需要发送的数据帧进行排序。

在作为接收方时，每个 CAN 控制器均提供两个 FIFO，每个 FIFO 可以存放 3 个 CAN 数据帧，并且其溢出级可以通过编程配置；提供 28 个可裁剪和配置的标识符过滤器槽，用来识别拥有不同标识符的 CAN 数据帧，以便判别丢弃或接收。

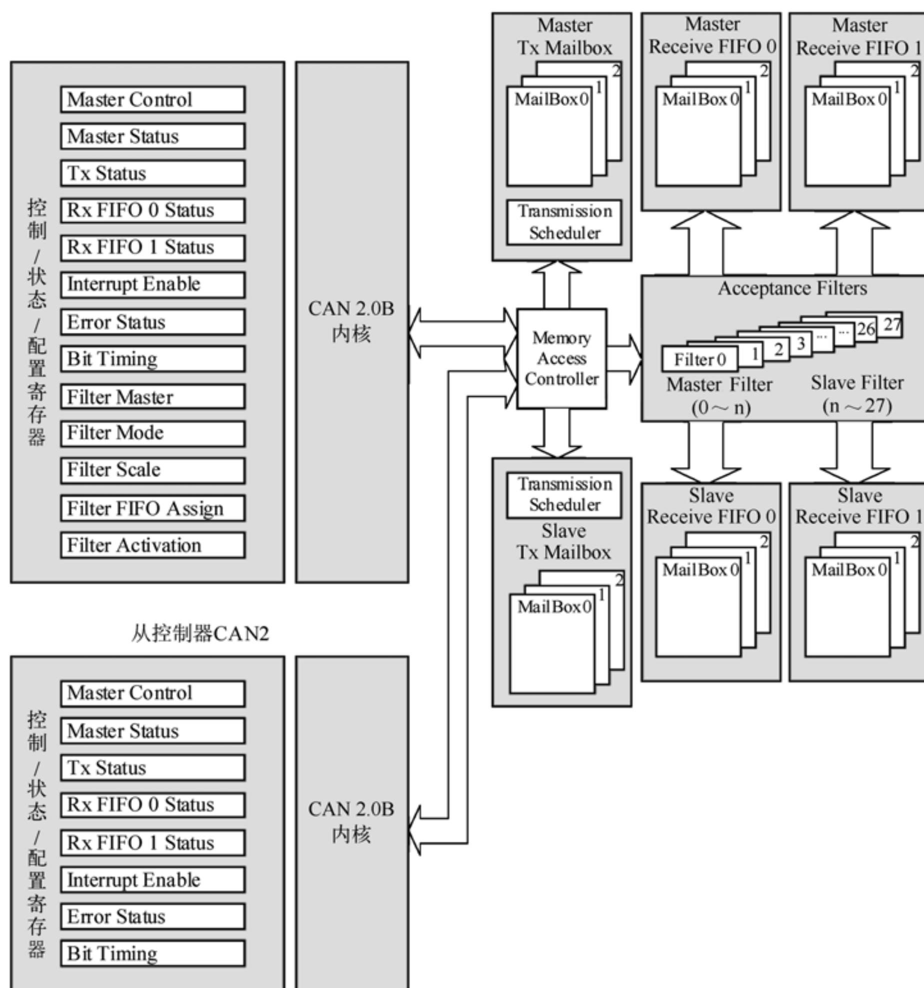


图 5-9　STM32F107 的 CAN 控制器结构框图

CAN 控制器的工作模式分为初始化模式、正常工作模式和睡眠模式，在硬件复位后，CAN 控制器处于睡眠模式以省电，同时 CANTX 引脚的内部上拉电阻处于使能状态。用户软件可以通过设置 CAN_MCR 寄存器中的 INRQ 或 SLEEP 位来使 CAN 控制器进入初始化模式或者睡眠模式；一旦 CAN 控制器进入初始化模式或者睡眠模式，会通过设置 CAN_MSR 寄存器中的 INAK 或 SLAK 位来确认状态变化，同时 CANTX 引脚的内部上拉电阻变为禁止状态。如果 CAN_MSR 寄存器中的 INAK 或 SLAK 位均未设置，则表明 CAN 控制器处于正常工作状态。

在初始化模式下，CAN 控制器会停止一切 CAN 通信活动，用户可以通过软件对 CAN 控制器进行初始化，包括配置位时间寄存器、控制寄存器和标识符过滤槽等。完成初始化后，用户可以通过清除 CAN_MCR 寄存器中的 INRQ 位并等待 CAN_MSR 寄存器中的 INAK 位清零来使控制器进入正常工作模式，CAN 控制器开始 CAN 总线数据帧的收发工作。为了省电，用户可以使 CAN 控制器进入睡眠模式，在睡眠模式下，CAN 控制器既可以由软件唤醒，也可以由 CAN 总线唤醒。

5.3.3 STM32F107 的 CAN 控制器操作

1. CAN 总线发送操作

要发送一个 CAN 总线数据帧，用户首先需要检查是否有处于空闲状态的发送邮箱，如果有，则将帧标识符、数据域长度及有效数据写入该发送邮箱的相应位置，并向 CAN_TIxR 寄存器中相应的 TXRQ 位写 1 请求 CAN 控制器发送数据帧。CAN 控制器在检查到 TXRQ 位为 1 后，会将该邮箱状态改为挂起状态，并根据优先级排队算法将其加入发送队列中的相应位置，一旦该邮箱成为优先级更高的挂起邮箱，该邮箱则进入调度状态，CAN 控制器将在物理总线空闲的时候将该邮箱中的数据帧发送出去（发送状态）。发送成功后，CAN 控制器将该发送邮箱的状态改为空闲，并通过将 CAN_TSR 寄存器中的 RQCP 和 TXOK 置位向用户指示发送成功。如果发送失败，CAN 控制器则将 CAN_TSR 寄存器中的 ALST 位置位，并/或将 TERR 位置位以指示发送错误。

当有多个发送邮箱需要执行发送操作时，CAN 控制器可以采用两种优先级排队算法对需要发送的数据帧进行优先级排队：

1）根据数据帧标识符进行优先级排队。CAN 控制器首先检查各个邮箱中数据帧的标识符，标识符小的先发送。如果两个邮箱中的数据帧标识符一样，则先发送编号小的邮箱中的数据。

2）根据发送请求的先后顺序进行优先级排队。要进入这种优先级排队方式，需要将 CAN_MCR 寄存器中的 TXFP 位置位。

如果用户想要放弃一个数据帧的发送，可以将 CAN_TSR 寄存器中相应的 ABRQ 位置位，当该发送邮箱处于挂起状态或者调度状态时，CAN 控制器将立即取消该邮箱的本次发送；当邮箱处于发送状态时，若本次发送已经成功，CAN 控制器将该邮箱状态设置为空闲，并将 CAN_TSR 寄存器中的 TXOK 位置位，若本次发送失败，CAN 控制器将该邮箱状态设置为空闲，并将 CAN_TSR 寄存器中的 TXOK 位清除。

2. CAN 总线接收操作

前面提到，STM32F107 中的每个 CAN 控制器都有两个 FIFO 用作数据接收。CAN 控制器将三个接收邮箱组成一个 FIFO，FIFO 的操作全部由硬件完成，不需要用户软件的干预，用户软件只需要根据 FIFO 的状态从 FIFO 的输出邮箱中读取数据即可，从而最大限度地节约了处理时间。

当 CAN 控制器接收到一个符合 CAN 总线协议的数据帧，并且该数据帧的标识符能够通过过滤器验证，CAN 控制器将该数据帧放入接收 FIFO 中。每个接收 FIFO 有 4 种状态，当 FIFO 中没有数据帧时，FIFO 处于空闲状态，若一个数据帧被放入 FIFO，则 FIFO 进入挂起状态 1，并将 CAN_RFR 寄存器中的 FMP[1 : 0]位设置为 01b 向用户软件指示该状态，同时，CAN 控制器将该数据帧放入 FIFO 的输出邮箱，供用户读取。用户从输出邮箱中读取了数据帧后，需要将 CAN_RFR 寄存器中的 RFOM 位置位来释放输出邮箱，这样，FIFO 又重新进入了空闲状态。如果在用户释放 FIFO 的输出邮箱的同时又收到一个新的数据帧，FIFO 将保持挂起状态 1，并将新的数据帧写入输出邮箱。

如果在用户未释放输出邮箱时又收到新的有效数据帧，FIFO 将进入挂起状态 2，CAN_RFR 寄存器中的 FMP[1 : 0]位变为 10b，如果再收到一个有效数据帧，FIFO 进入挂起状态 3，CAN_RFR 寄存器中的 FMP[1 : 0]位变为 11b。这时，用户必须立即读取输出邮箱中的

数据帧并释放邮箱，否则如果再收到一个有效数据帧，CAN 控制器会设置 CAN_RFR 寄存器中的 FOVR 位，指示 FIFO 发生了溢出并丢失了数据帧——若 FIFO 的锁定功能未使能（CAN_MCR 寄存器中的 RFLM 位清零），FIFO 中最早收到的数据帧将被最新收到的数据帧取代；若锁定功能使能（CAN_MCR 寄存器中的 RFLM 位置位），则 FIFO 中的内容不变，最新收到的数据帧被丢弃。

3. 标识符过滤

为了实现 CAN 总线数据帧的标识符过滤，STM32F107 中的每个 CAN 控制器将提供 28 个可配置可裁剪的标识符过滤槽，这些标识符过滤槽可以不需要 CPU 的参与，自动完成标识符的自动过滤，只接收具有用户指定标识符的 CAN 总线数据帧。

每个标识符过滤槽可以单独配置宽度，既可以配置为一个 32bit 的过滤器以对应 STDID[10∶0]、EXTID[17∶0]、IDE 和 RTR.，也可以配置为两个 16bit 过滤器以对应 STDID[10∶0]、RTR、IDE 和 EXTID[17∶15]，如图 5-10 所示。标识符过滤槽有两种工作模式，分别为屏蔽模式和标识符列表模式。在屏蔽模式下，标识符槽中的标识符寄存器和屏蔽寄存器配合，共同决定标识符的某个位是“必须匹配”还是“不关心”。在标识符列表模式下，标识符槽中的屏蔽寄存器也作为标识符寄存器使用。

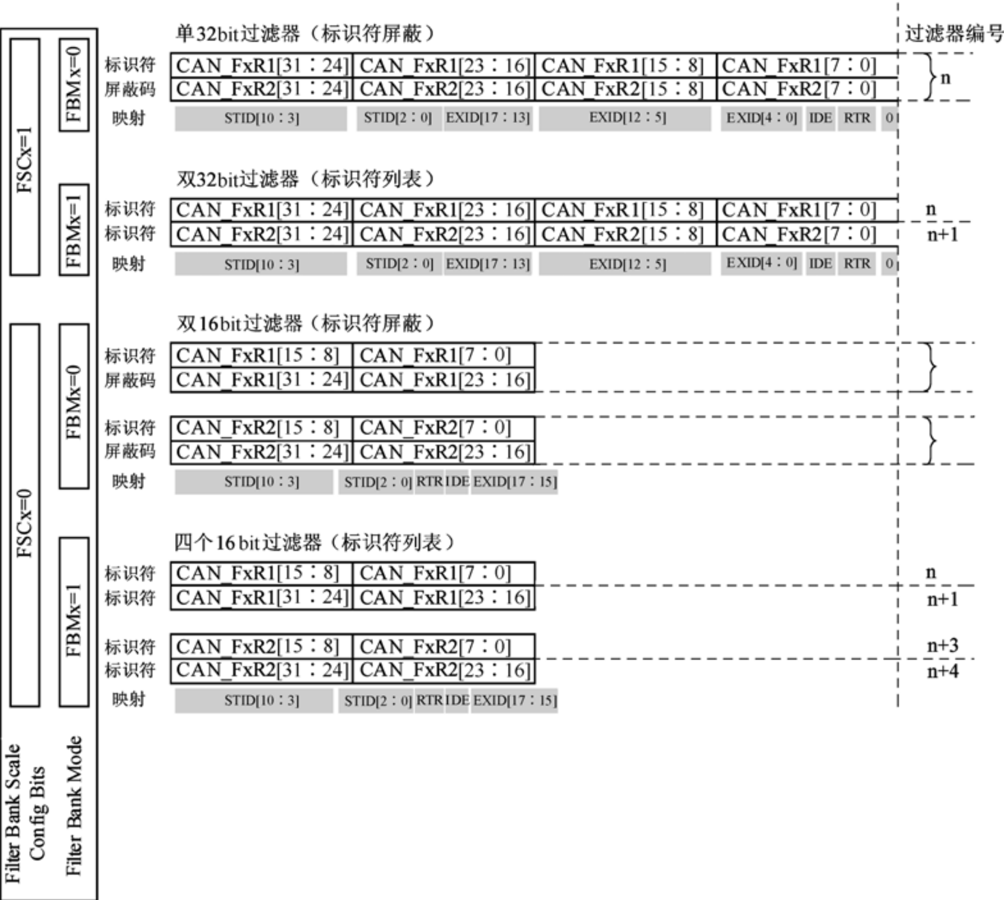


图 5-10　标识符槽的裁减配置及寄存器组织图

标识符过滤槽的配置主要通过对相应的 CAN_FMR 寄存器中 FSCx 位和 FBMx 位的设置来完成，请参照图 5-10。在配置过滤槽之前，必须把 CAN_FAR 寄存器中的 FACT 位清零，以保证该过滤槽处于未激活状态。

4. 存储器组织

STM32F107 中的 CAN 控制器是通过邮箱的方式完成硬件与用户软件之间的数据交换的，邮箱中包含了一个 CAN 总线数据帧的所有信息，包括标识符、数据、控制信息、状态信息和时间戳等。

在发送时，用户写入邮箱相应位置的信息将由硬件写入到 CAN 控制器的相应寄存器，见表 5-18。发送状态有在 CAN_TSR 寄存器中指示。

表 5-18　CAN 控制器的相应寄存器

相对于发送邮箱基地址的偏移量	寄存器名称
0	CAN_TIxR
4	CAN_TDTxR
8	CAN_TDLxR
12	CAN_TDHxR

在接收时，用户通过读取 FIFO 的输出邮箱获得数据帧，邮箱的相应位置存放着收到该数据帧时 CAN 控制器相应的寄存器，见表 5-19。

表 5-19　邮箱存放着收到数据帧时 CAN 控制器相应的寄存器

相对于 FIFO 输出邮箱基地址的偏移量	寄存器名称
0	CAN_RIxR
4	CAN_RDTxR
8	CAN_RDLxR
12	CAN_RDHxR

5.3.4　基于 STM32F107 的 CAN 硬件设计

基于 STM32F107 的 CAN 总线硬件设计与基于 C8051F500 的 CAN 总线硬件设计类似，需要外接总线驱动芯片 PCA82C250。图 5-11 是 CAN 总线接口电路的电路图，CAN1TX1 对应 STM32F107 CAN1 控制器的发送引脚，CAN1RX1 对应 STM32F107 CAN1 控制器的接收引脚。为了避免 CAN 总线可能对系统带来的干扰和损害，采用高速光耦 HCPL-0601 进行了总线隔离，+5V-CAN 和 GND-CAN 是经过隔离后的 5V 电源和地。为了进一步提高系统的可靠性，在 CAN 总线 CANH 和 CANL 间增加了瞬态抑制器 LCDA15 对总线进行保护。

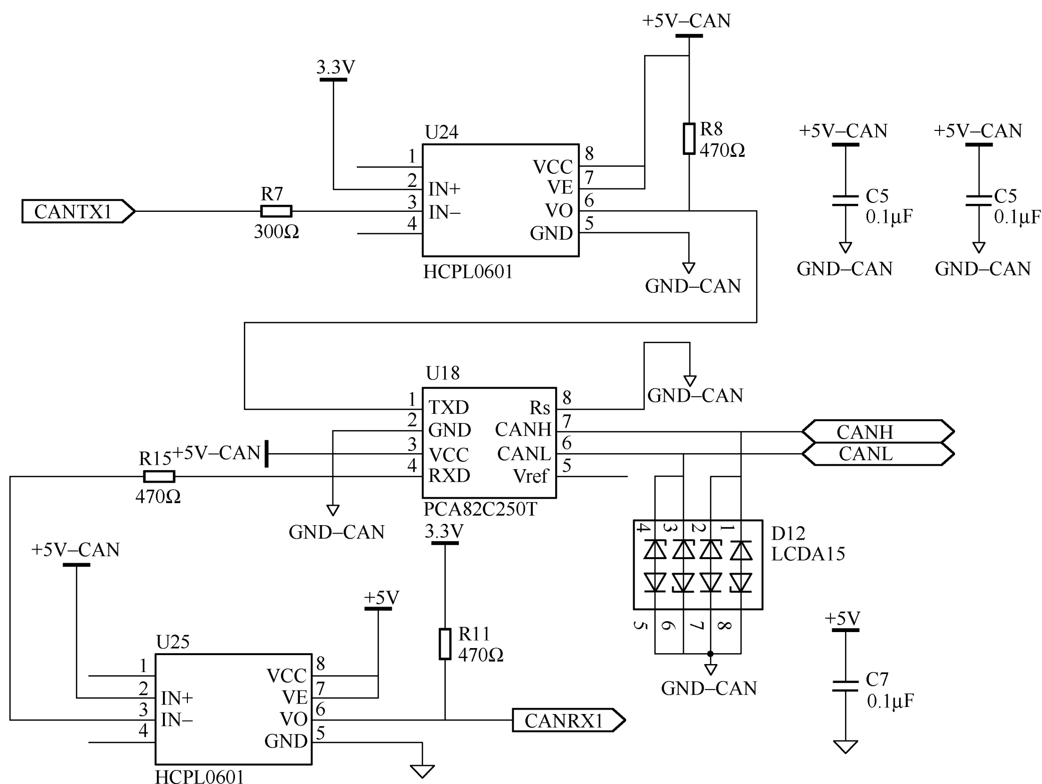


图 5-11　CAN 总线接口电路

5.3.5　基于 STM32F107 的 CAN 软件设计

意法半导体公司为 STM32 系列提供了丰富的软件库，其中就包含了 CAN 控制器的应用库，编程非常简单，一个简单的初始化程序代码如下：

```
Void Init_CAN(void)
{
    CAN_InitTypeDef              CAN_InitStructure;
    CAN_FilterInitTypeDef   CAN_FilterInitStructure;
    //配置系统时钟
    CanRCC_Configuration( );
    //配置 IO 管脚，使相应的 IO 管脚作为 CAN 收发使用
    CanGPIO_Configuration( );
    //配置 CAN 中断
    CanNVIC_Configuration( );
    //配置 CAN 寄存器
    CAN_DeInit(CAN1);
    CAN_StructInit(&CAN_InitStructure);
    CAN_InitStructure.CAN_TTCM=DISABLE;
    CAN_InitStructure.CAN_ABOM=DISABLE;
    CAN_InitStructure.CAN_AWUM=DISABLE;
```

```
        CAN_InitStructure.CAN_NART=DISABLE;
        CAN_InitStructure.CAN_RFLM=DISABLE;
        CAN_InitStructure.CAN_TXFP=DISABLE;
        CAN_InitStructure.CAN_Mode=CAN_Mode_Normal;
        CAN_InitStructure.CAN_SJW=CAN_SJW_1tq;
        CAN_InitStructure.CAN_BS1=CAN_BS1_10tq;
        CAN_InitStructure.CAN_BS2=CAN_BS2_7tq;
        CAN_InitStructure.CAN_Prescaler=4;
        CAN_Init(CAN1,&CAN_InitStructure);
        //初始化 CAN 过滤器
        CAN_FilterInitStructure.CAN_FilterNumber=0;
        CAN_FilterInitStructure.CAN_FilterMode=CAN_FilterMode_IdMask;
        CAN_FilterInitStructure.CAN_FilterScale=CAN_FilterScale_32bit;
        CAN_FilterInitStructure.CAN_FilterIdHigh=0x0000;
        CAN_FilterInitStructure.CAN_FilterIdLow=0x0000;
        CAN_FilterInitStructure.CAN_FilterMaskIdHigh=0x0000;
        CAN_FilterInitStructure.CAN_FilterMaskIdLow=0x0004;
        CAN_FilterInitStructure.CAN_FilterFIFOAssignment=0;
        CAN_FilterInitStructure.CAN_FilterActivation=ENABLE;
        CAN_FilterInit(&CAN_FilterInitStructure);

        CAN_FilterInitStructure.CAN_FilterNumber=15;
        CAN_FilterInitStructure.CAN_FilterMode=CAN_FilterMode_IdMask;
        CAN_FilterInitStructure.CAN_FilterScale=CAN_FilterScale_32bit;
        CAN_FilterInitStructure.CAN_FilterIdHigh=0x0000;
        CAN_FilterInitStructure.CAN_FilterIdLow=0x0000;
        CAN_FilterInitStructure.CAN_FilterMaskIdHigh=0x0000;
        CAN_FilterInitStructure.CAN_FilterMaskIdLow=0x0004;
        CAN_FilterInitStructure.CAN_FilterFIFOAssignment=0;
        CAN_FilterInitStructure.CAN_FilterActivation=ENABLE;
        CAN_FilterInit(&CAN_FilterInitStructure);
        //使能 CAN1 FIFO0  的接收中断
        CAN_ITConfig(CAN1,CAN_IT_FMP0, ENABLE);
    }
```

CAN1 FIFO0 的接收中断程序：

```
    void CAN1_RX0_IRQHandler(void)
    {
        CanRxMsg RxMessage;
        RxMessage.StdId=0x00;
        RxMessage.ExtId=0x00;
        RxMessage.IDE=0;
        RxMessage.DLC=0;
        RxMessage.FMI=0;
        CAN_Receive(CAN1,CAN_FIFO0, &RxMessage);
    }
```

CAN 发送子程序：

```
void CAN_tx(u8 *data)
{
        CanTxMsg TxMessage;
        u8 TransmitMailbox1;
        u8 datap;
        int j;
        //  写入基本的帧信息：标准数据帧，数据长度为 8B
        TxMessage.RTR=CAN_RTR_DATA;
        TxMessage.IDE=CAN_ID_STD;
        TxMessage.DLC=8;
        //从发送缓冲区的头两个字节中获取帧标识符
        datap = 0;
        TxMessage.StdId = (data[datap] | data[datap+1] << 8);
            datap += 2;
            //从发送缓冲区中获取有效数据
          for (j=0; j<8; j++){
                TxMessage.Data[j]=data[datap++];
          }
          //执行发送
          TransmitMailbox1=CAN_Transmit(CAN1,&TxMessage);
          //延迟 1ms
          vTaskDelay(1);
          //如果未发送成功，则放弃本次发送
                if (CAN_TransmitStatus(CAN1,TransmitMailbox1) != CANTXOK)
                      CAN_CancelTransmit(CAN1, TransmitMailbox1);
}
```

5.4 本章小结

本章介绍的几种带 CAN 总线接口的微处理器，这种方案使 CAN 总线应用电路设计更简单、更可靠，且使用灵活、方便。本章介绍了 3 种典型微处理的硬件、软件设计，均在科研实践中得到验证，可供大家参考。

思考题与习题

5.1 C8051F50X 系列单片机包含哪些 CAN 寄存器，这些寄存器采用何种方式进行访问？

5.2 简述 C8051F0X 系列单片机的 CAN 控制器的初始化步骤？

5.3 TMS320LF28335 的 32 个邮箱是如何控制和分配的？

5.4 系统时钟为 150MHz 时，如何配置 TMS320LF28335 的位定时配置寄存器，使得 eCAN 的位速率为 500kbit/s？

5.5 STM32F107 的两个 CAN 控制器的存储器是如何组织的？

5.6 STM32F107 的 CAN 标识符过滤槽有几种工作模式？有何区别？

第 6 章　CAN 总线与计算机的接口设计

计算机一般无法与 CAN 总线直接连接，需要设计计算机专用的 CAN 总线接口卡。本章介绍了以下 6 种基于不同计算机总线的 CAN 接口卡，给出了详细的硬件和软件资料。

本章要点

- PC-104 总线 CAN 接口卡设计；
- ISA 总线 CAN 接口卡设计；
- PCI 总线 CAN 接口卡设计；
- PC 并行端口与 CAN 接口设计；
- USB 端口与 CAN 接口设计；
- 以太网端口与 CAN 接口设计。

6.1　PC-104 总线 CAN 接口卡设计

6.1.1　PC-104 总线介绍

1．PC-104 总线概述

PC-104 工控机采用独特的叠式总线结构，其总线之间的连接使用了 104 根信号线，可分成 5 类：地址线、数据线、控制线、时钟线、电源线。由于 PC-104 与 PC 软件、硬件完全兼容，所以可为技术人员提供良好的开发环境，并且 PC-104 模块体积小，可方便构成嵌入式计算机应用系统。

（1）地址线

SA0～SA19 和 LA17～LA23，其中 SA0～SA19 是可锁存的地址信号，LA17～LA23 为非锁存信号，由于没有锁存延时，所以给外设插板提供了一条快捷途径。SA0～SA19 加上 LA17～LA23 可实现 16MB 空间寻址（其中，SA13～SA19 和 LA17～LA23 是重复的）。

（2）数据线

SD0～SD7 和 SD8～SD15，其中数据线 SD0～SD7 为低 8bit 数据，SD8～SD15 为高 8bit 数据。

（3）控制线

AEN：地址允许信号，输出线，高电平有效。AEN=1，表明处于 DMA 控制周期；AEN=0，表示非 DMA 周期。此信号用来在 DMA 期间禁止 I/O 端口的地址译码。

BALE：允许地址锁存，输入线，此信号由总线控制器 8288 提供，作为 CPU 地址的有效标志，当 BALE 为高电平时，将 SA0～SA19 接到系统总线，其下降沿用来锁存 SA0～SA19。

$\overline{\text{IOR}}$：I/O 读命令，输出线，低电平有效，用来把选中的 I/O 设备的数据读到数据总线上。在 CPU 启动的 I/O 周期，通过地址线选择 I/O；在 DMA 周期，I/O 设备由 DACK 选择。

$\overline{\text{IOW}}$：I/O 写命令，输出线，低电平有效，用来把数据总线上的数据写入被选中的 I/O 端口。

$\overline{\text{SMEMR}}$ 和 $\overline{\text{SMEMW}}$：存储器读/写命令，低电平有效，用于对 SA0～SA19 这 20bit 地址寻址的 1MB 内存的读/写操作。

$\overline{\text{MEMR}}$ 和 $\overline{\text{MEMW}}$：低电平有效，存储器读/写命令，用于对 SA0～SA19 这 20bit 地址寻址的 1MB 内存的读/写操作。

$\overline{\text{MEMCS16}}$ 和 $\overline{\text{IOCS16}}$：它们是存储器 16bit 片选信号和 I/O 外设 16bit 片选信号，分别指明当前数据传送的是 16bit 存储器周期还是 16bit I/O 周期。

SBHE：总线高字节允许信号，该信号有效时，表示数据总线上传送的是高字节数据。

IRQ3～IRQ7 和 IRQ10～IRQ15：用于作为外部设备的中断请求输入线，分别连到主片 8259A 和从片 8259A 中断控制器的输入端，其中，IRQ13 留给数据协处理器使用，不在总线上出现。这些中断请求线都是边沿触发，三态门驱动器驱动。优先级排队是 IRQ0 最高，依次为 IRQ1，IRQ8～IRQ15，IRQ3～IRQ7。

DRQ0～DRQ3 和 DRQ5～DRQ7：来自外部设备的 DMA 请求输入线，高电平有效，分别连到主片 8237A 和从片 8237A DMA 控制器输入端。DRQ0 优先级最高，DRQ7 最低，DRQ4 用于级联，在总线上不出现。

$\overline{\text{DACK0}}$～$\overline{\text{DACK3}}$ 和 $\overline{\text{DACK5}}$～$\overline{\text{DACK7}}$：DMA 应答信号，低电平有效。有效时，表示 DMA 请求被接收，DMA 控制器占用总线，进入 DMA 周期。

T/C：DMA 终末/计数结束，输出线。该信号是一个正脉冲，表明 DMA 传送的数据已达到其程序预置的字节数，用来结束一次 DMA 数据块传送。

$\overline{\text{MASTER}}$：输入信号，低电平有效。它由要求占用总线的有主控能力的外设卡驱动，并与 DRQ 一起使用。外设的 DRQ 得到确认（DACK 有效）后，才使 MASTER 有效，从此该设备保持对总线的控制直到 $\overline{\text{MASTER}}$ 无效。

RESETDRV：系统复位信号，输出线，高电平有效。此信号在系统电源接通时为高电平，当所有电平都达到规定后变低，即上电复位时有效。用它来复位和初始化接口和 I/O 设备。

$\overline{\text{I/OCHCK}}$：I/O 通道检查，输出线，低电平有效。当它为低电平时，表明接口插件的 I/O 通道出现了错误，它将产生一次不可屏蔽中断。

I/OCHRDY：I/O 通道就绪，输入线，高电平表示“就绪”。该信号可供低速 I/O 设备或存储器请求延长总线周期之用。当低速设备被选中，且收到读或写命令时将此线电平拉低，表示未就绪，以便在总线周期中加入等待状态 TW，但最多不能超过 10 个时钟周期。

$\overline{\text{OWS}}$：零等待状态信号，输入线。该信号为低电平时，无需插入等待周期。

2．PC-104 总线 8bit I/O 工作时序

PC-104 总线工作时序分为访问存储器和访问端口两种。图 6-1 为 PC-104 总线 8bit I/O 工作时序图。如图 6-1 所示，在 BALE 上升沿时，端口地址被锁存在 SA 总线上。从设备可在 BALE 的下降沿采样该地址，SA 总线上的端口地址在整个 I/O 周期结束之前一直保持有效。在整个 I/O 周期间，AEN 始终为低电平。当 $\overline{\text{IOR}}$ 或 $\overline{\text{IOW}}$ 有效时，CPU 可对指定 I/O 端口进行相应的 I/O 读写操作。在读操作期间，总线上数据在 $\overline{\text{IOR}}$ 信号上升沿时必须有效。在写操作期间，总线上数据在整个 $\overline{\text{IOW}}$ 为低电平期间一直有效。

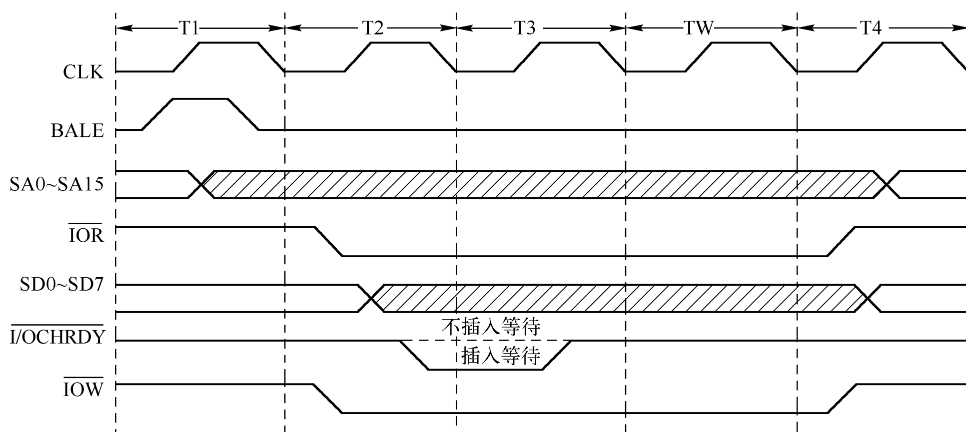


图 6-1　PC-104 总线 8bit I/O 工作时序

6.1.2　硬件电路设计说明

硬件接口电路主要分为基地址设置、I/O 读写接口、中断接口、复位信号、CAN 输出接口几个部分。

1．基地址设置

一般 PC 在进行 I/O 读写操作时只使用了 SA0～SA9 十根地址线作为寻址线使用，此卡使用了 SA2～SA9 八根地址线作为基地址（BASE_ADDRESS）译码信号，通过与卡上预设的地址相比较输出基地址有效信号 $\overline{\text{CS}}$，如图 6-2 所示。SA0、SA1 作为卡内地址参与译码，共使用了从基地址开始的连续 4 个 I/O 地址。

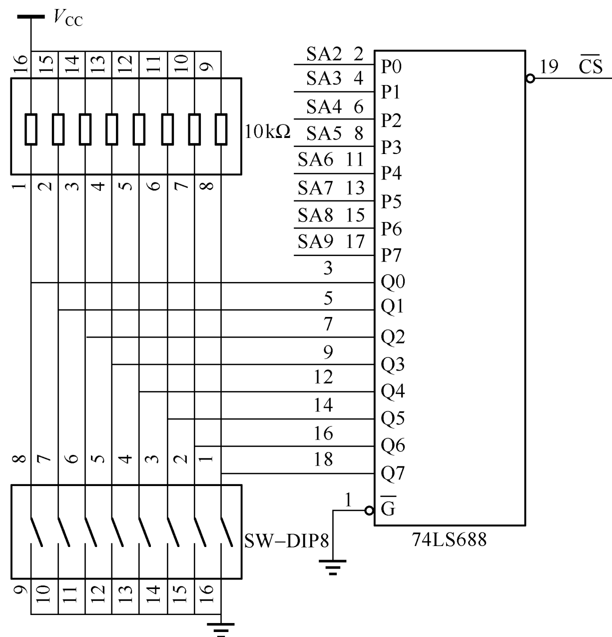


图 6-2　基地址设置电路

2．I/O 读写接口

PC-104 总线的地址和数据线是分离的，而 SJA1000 的地址和数据是复用的，所以无法直接连接。可以使用两个连续的 PC-104 总线的 I/O 操作合成一个对 SJA1000 的 I/O 操作，即第一次 I/O 周期通过数据线向 SJA1000 写入片内寄存器地址，第二次 I/O 周期从 SJA1000 写入（读出）数据。

此卡是双 CAN 卡，有两片 SJA1000，编为 1 号、2 号，需要的读写信号有：

$\overline{\text{CS1}}$:1 号 SJA1000 片选信号；$\overline{\text{CS2}}$:2 号 SJA1000 片选信号；ALE:地址锁存信号；$\overline{\text{WR}}$ 写信号；$\overline{\text{RD}}$:读信号。

同时要避免 PC 在进行 DMA 操作时可能发生的地址冲突。信号的逻辑关系如图 6-3 所示。$\overline{\text{RD}}$ 与 $\overline{\text{I/OW}}$ 直接相连。

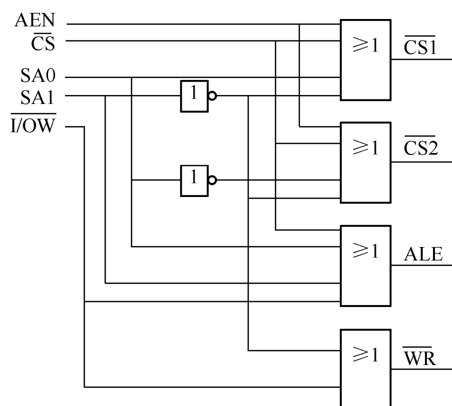


图 6-3　信号的逻辑关系

要完成一次读/写操作，首先向（BASE_ADDRESS+0）端口写入片内寄存器地址，再从（BASE_ADDRESS+2）端口（1 号）读出/写入数据，或者从（BASE_ADDRESS+3）端口（2 号）读出/写入数据。进行 I/O 读写的时序，如图 6-4 所示。在 PC 进行 DMA 操作时 AEN 有效，使 $\overline{\text{CS1}}$、$\overline{\text{CS2}}$ 无效。

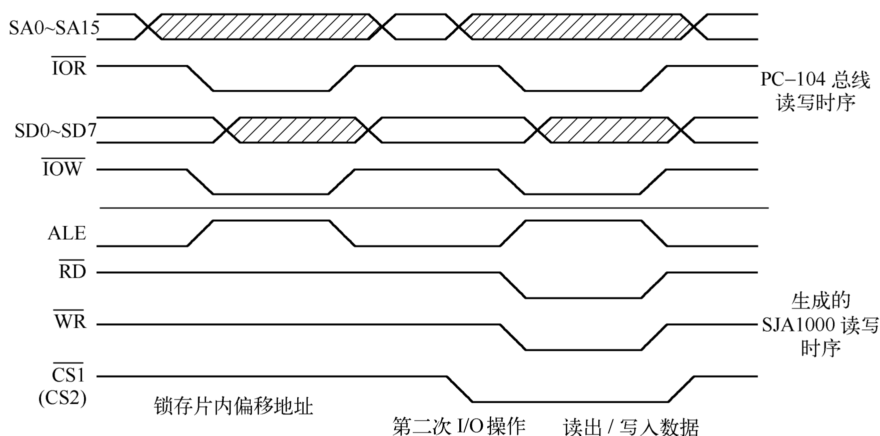


图 6-4　I/O 读写时序图

3．中断接口

PC-104 总线的中断输入信号 IRQx 为高电平有效，SJA1000 的中断输出信号 $\overline{\text{CAN-INT}}$ 为低电平有效。二者的接口需要一个非门连接，为了节省 PC 的资源，可以通过短接器短接两个 SJA1000 的中断输出，使其共用一个中断，如图 6-5 所示。同时考虑到保持与 PC 其他中断资源的线或关系，采用了漏开输出的非门 7406。

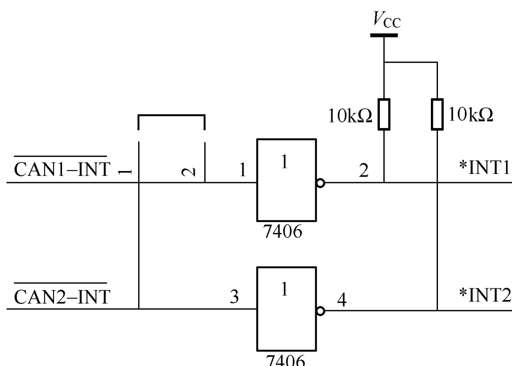


图 6-5　中断控制电路

4．复位信号

SJA1000 的复位输入信号为 $\overline{\text{RESET1}}$（$\overline{\text{RESET2}}$），PC-104 总线的上电复位输出信号为 *RESET，接入两个上升沿锁存的 D 触发器 7474，实现复位接口，如图 6-6 所示（注：信号前有*号的表示为 PC-104 总线信号，以下同）。SJA1000 在以下三种情况下复位：

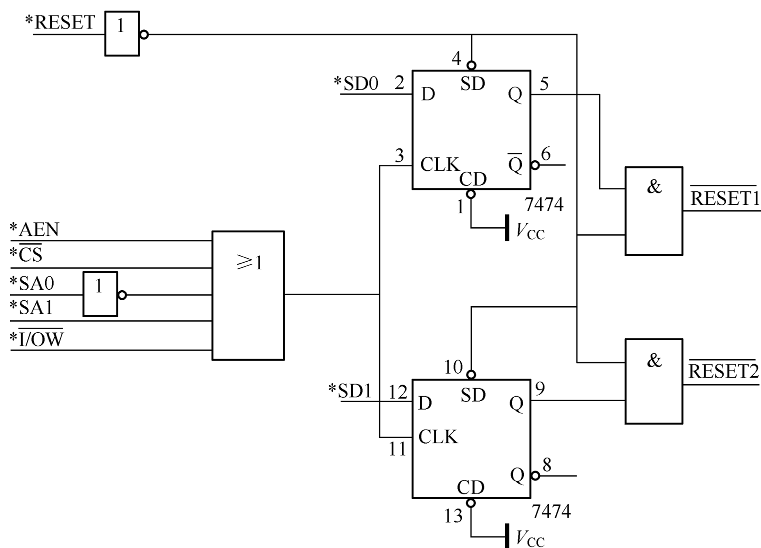


图 6-6　复位电路

① PC 启动上电时，两个 SJA1000 均复位；

② 对 1 号 SJA1000：向（BASE_ADDRESS+1）端口写入数据 0x02，延时一段时间，再写入 0x03；

③ 对 2 号 SJA1000：向（BASE_ADDRESS+1）端口写入数据 0x01，延时一段时间，再写入 0x03。

通信卡上逻辑电路是用一片 GAL16V8 实现的。

PC-104 总线与 SJA1000 之间的连接方式（见图 6-7）。

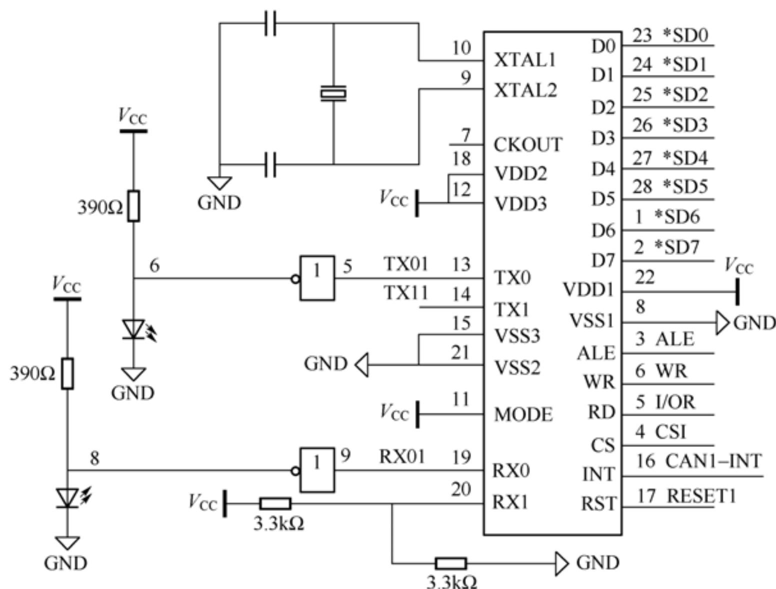


图 6-7　SJA1000 的连接方式

CAN 网络输出接口的硬件设计采取了光耦隔离的保护模式，并使用 DC/DC 电源模块实现电源隔离，如图 6-8 所示。

图 6-8　光耦隔离输出

为了提高通信卡的抗干扰和保护能力，在总线输出端增加了限流电阻和吸收电容等器件，同时通信卡内置终端电阻，通过跳线选择使用，如图 6-9 所示。

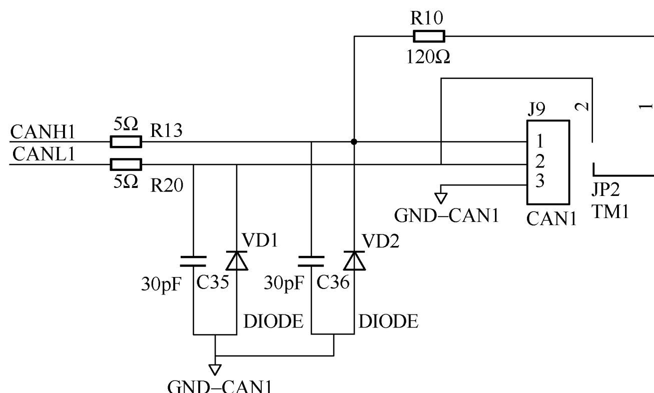


图 6-9　输出端保护电路

PC-104 双 CAN 接口卡的总电路原理图如图 6-10 所示。

6.1.3　PC–104 接口卡软件设计

主要介绍在 DOS 操作系统环境下的接口编程，程序分为对 SJA1000 的读写、SJA1000 的初始化、CAN 报文发送、CAN 报文接收、定时检测几个部分。

1．PC 对 SJA1000 的读写

根据硬件的设计可知，对 SJA1000 的读/写操作是通过两个 I/O 操作周期完成的。首先进行一次写操作，向 SJA1000 写入偏移地址；然后进行读/写操作，读出或写入数据。

定义基地址：#define Basic_Address 0x220。

（1）向 SJA1000 写入数据

入口参数：

```
num             —选择 SJA1000(1-1 号 SJA1000，2-2 号 SJA1000)
in_adress       —SJA1000 内部寄存器地址
out_data        —要写入的数据
void SJA_Write(int num, unsigned char in_adress，unsigned char out_data)
{
    outportb(Basic_Adress, in_adress);              //写入片内寄存器地址
    if(num==1)outportb(Basic_Adress+2, out_data);   //向 1 号 SJA1000 写入数据
    if(num==2)outportb(Basic_Adress+3, out_data);   //向 2 号 SJA1000 写入数据
}
```

（2）从 SJA1000 读出数据

入口参数：

```
num             —选择 SJA1000（1-1 号 SJA1000，2-2 号 SJA1000）
in_adress       —SJA1000 内部寄存器地址
```

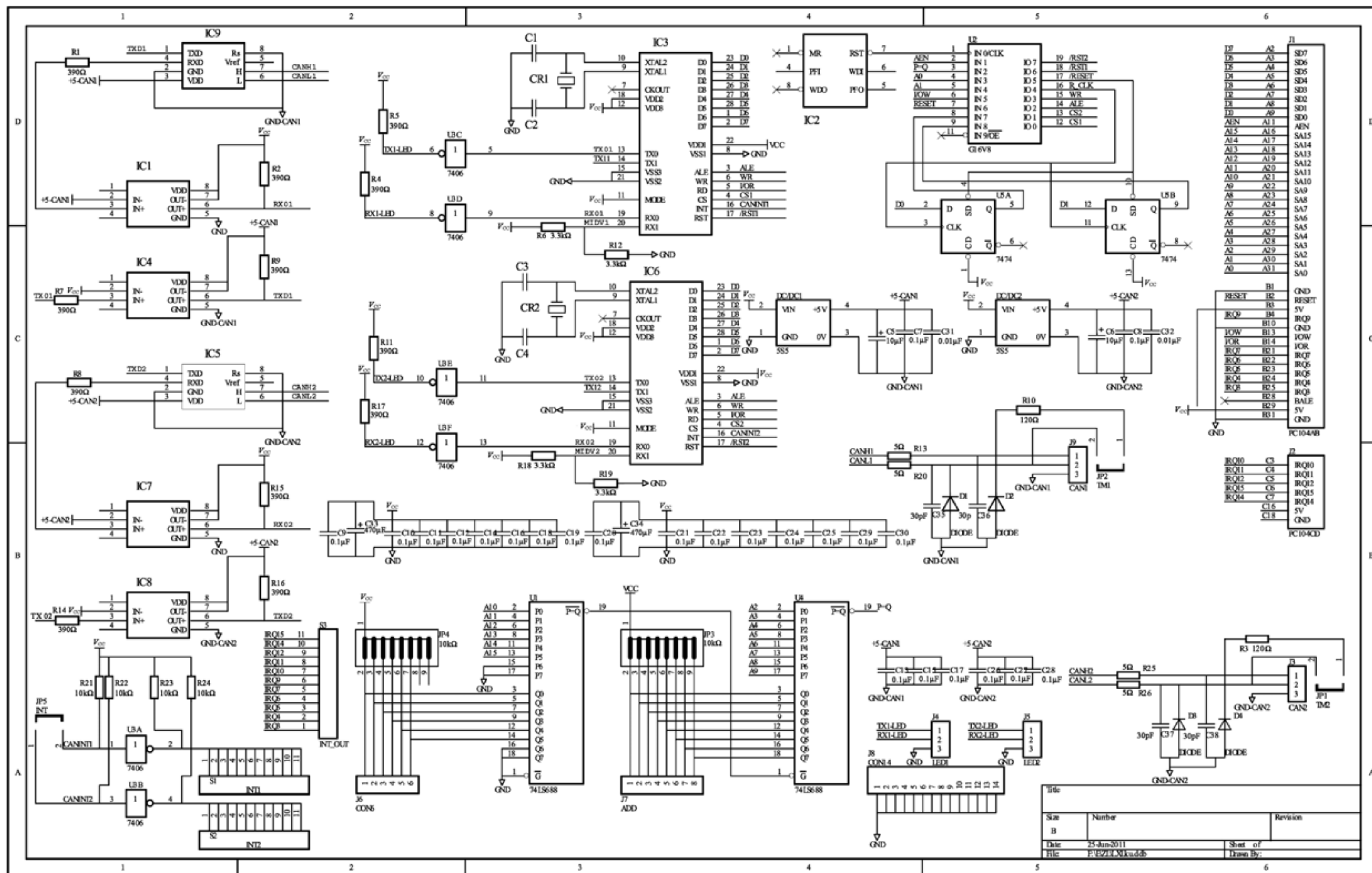


图 6-10　PC-104 接口卡电路原理图

返回值：读出的数据。

```
unsigned char SJA_Read(int num, unsigned char in_adress)
{
      unsigned char in_data;
      outportb(Basic_Adress, in_adress);                    //写入片内寄存器地址
      if(num==1) in_data=inportb(Basic_Adress+2);          //从 1 号 SJA1000 读出数据
      if(num==2) in_data=inportb(Basic_Adress+3);          //从 2 号 SJA1000 读出数据
      return(in_data);                                      //返回数据
}
```

2．CAN 适配卡初始化

对 CAN 适配卡的初始化主要是对 SJA1000 的初始化。初始化工作包括：工作方式的设置、接收滤波方式的设置、接收屏蔽寄存器（AMR）和接收代码寄存器（ACR）的设置、波特率参数和中断允许寄存器（IER）的设置等。

```
void initial_can( )                                         //初始化 SJA1000
{
       disable( );                                          //关中断
              //initial can1
              SJA_Write(1, 0, 0x01);

              SJA_Write(1, 4, Local_1);                     //验收 ID
              SJA_Write(1, 5, 0x00);
              SJA_Write(1, 6, 0x47);                        //波特率为 50kbit/s
              SJA_Write(1, 7, 0x2f);
              SJA_Write(1, 8, 0xda);                        //推挽输出
              SJA_Write(1, 0, 0x72);
              //initial can2
              SJA_Write(2, 0, 0x01);
              SJA_Write(2, 4, Local_2);                     //验收 ID
              SJA_Write(2, 5, 0x00);
              SJA_Write(2, 6, 0x47);                        //波特率为 50kbit/s
              SJA_Write(2, 7, 0x2f);
              SJA_Write(2, 8, 0xda);                        //推挽输出
              SJA_Write(2, 0, 0x72);
              enable( );                                    //开中断
       }
```

3．CAN 报文发送

报文的发送函数负责将用户要发送的数据写入 SJA1000 发送缓存寄存器，然后启动发送。

入口参数：

```
num                 -选择 SJA1000(1-1 号 SJA1000，2-2 号 SJA1000);
aim_adress          -目标地址；
out_data            -要发送的数据。
void send(int num, unsigned char aim_adress, unsigned char out_data)
```

```
{
        unsigned char temp;
        temp=SJA_Read(num, 2);
        if((temp&0x04)==0x04)                          //TBF 是否可写，是，则写入报文
        {
                SJA_Write(num, 10, aim_adress);        //写报文 ID
                SJA_Write(num, 11, 1);                 //写发送字节数
                SJA_Write(num, 12, out_data);          //写发送数据
                SJA_Write(num, 1, 0x01);               //启动发送
        }
}
```

4．CAN 报文接收

接收采用中断方式，当有数据到达时触发 PC 中断，然后从 SJA1000 的接收缓冲器读入数据。程序分为装载中断和中断程序两部分。

定义中断向量号：#define Int_num 0x0a　//中断 IRQ9 0x0a

装载中断：

```
void install_inter( )
{
        disable( );                                    //关中断
        old_can_inter=getvect(Int_num);                //保存旧中断入口
        setvect(Int_num, can_inter);                   //设置新中断入口
        enable( );                                     //开中断
}
```

中断程序：

```
void interrupt can_inter( )                            //CAN 接收中断
{
        unsigned char temp;
        disable( );                                    //关中断
        //can1 interrupt
        temp=SJA_Read(1, 3);
        if(temp&0x08)                                  //是否超载
        {
                SJA_Write(1, 1, 0x08)                  //是，则清除超载状态
        }
        if(temp&0x01)                                  //若有接收中断
        {
                canbuf[1]=SJA_Read(1, 22);             //则将数据写入缓冲区
                SJA_Write(1, 1, 0x0c);                 //释放 TBF
                putchar(canbuf[1]);                    //显示数据
                puts("=>Can1 Receive");
        }
          outportb(0x20, 0x20);                        //发中断结束命令
          enable( );                                   //开中断
}
```

5．定时检测

为了提高系统的可靠性，需要定时对 SJA1000 的状况进行检测。如果发现其脱离总线就要做复位处理。定时检测程序是通过定时中断的方法实现的，分为定时中断安装和执行两部分。

定义中断向量号：#define Int_time 0x1c　　　　　//定时中断向量号

定时中断的安装：

```
void install_inter( )                          //装载中断
{
    disable( );                                    //关中断
    old_time_inter=getvect(Int_time);          //保存旧中断入口
    setvect(Int_time，time_inter);             //设置新中断入口
    enable( );                                     //开中断
}
```

定时检测程序：

```
void interrupt time_inter( )                   //定时检测 SJA1000 是否总线脱离
{
    unsigned char temp;
    disable( );                                //关中断
    temp=SJA_Read(1, 2);
    if(temp&0x80)==0x80)
    {
        SJA_Write(1, 0, 0x01);                 //若脱离，则将 SJA1000 复位
        SJA_Write(1, 0, 0x72);                 //在进入正常工作方式
    }
    temp=SJA_Read(2, 2);
    if((temp&0x80)==0x80)
     {
         SJA_Write(2, 0, 0x01);                //若脱离，则将 SJA1000 复位
         SJA_Write(2, 0, 0x72);                //在进入正常工作方式
     }
     enable( );                                //开中断
}
```

6．PC-104 双 CAN 接口卡源程序

```
/*********************************************************************************
    基于 PC-104 总线的双 CAN 网通信卡的配套测试程序
    编程语言：C 语言
    运行环境：DOS
    测试功能：测试同一块卡上的两个 SJA1000 之间的通信。
    注意事项：测试前将通信板的基地址设置为 0x220，中断设置为共用 IRQ9，
              通过外部连接线连接卡内的两路 CAN 总线，短接 TM1、TM2。
    设置完成后运行测试程序，根据软件的提示操作即可。
*********************************************************************************/
```

```
#include "stdio.h"
#include "dos.h"

#define Basic_Adress 0x220                          //基地址
#define Int_num 0x0a                                //中断向量号 IRQ9 0x0a
#define Int_time 0x1c                               //定时中断向量号
#define   Local_1    1                              //1 号 CAN 的本站地址
#define   Local_2    2                              //2 号 CAN 的本站地址

void interrupt can_inter( );                        //CAN 接收中断
void interrupt time_inter( );                       //定时中断
void interrupt (*old_can_inter)( );                 //保存旧中断入口
void interrupt (*old_time_inter)( );
void install_inter( );                              //装载中断
void resume_inter( );                               //恢复中断
void initial_can( );                                //初始化 SJA1000

/*写 SJA1000，参数：       num-选择 SJA1000(1-1 号 SJA1000，2-2 号 SJA1000);
                          in_adress-SJA1000 内部寄存器地址
                          out_data-要写入的数据*/
void SJA_Write(int num,unsigned char in_adress,unsigned char out_data);

/*读 SJA1000，参数：       num-选择 SJA1000(1-1 号 SJA1000，2-2 号 SJA1000);
                          in_adress-SJA1000 内部寄存器地址
                          返回值-读出的数据*/
unsigned char SJA_Read(int num,unsigned char in_adress);

/*发送数据，参数：         num-选择 SJA1000(1-1 号 SJA1000，2-2 号 SJA1000);
                          aim_adress-目标地址
                          out_data-要发送的数据*/
void send(int num,unsigned char aim_adress,unsigned char out_data);

//复位 SJA1000，参数：num-选择 SJA1000(1-1 号 SJA1000，2-2 号 SJA1000)
void reset_can(int num);

void help( );      //帮助信息

unsigned char canbuf[5],inter_ocw;        //全局变量：canbuf-输入缓存；inter_ocw-中断屏蔽字

main( )
{
     unsigned char u_data;
     char in_key;
     unsigned long i;
     initial_can( );                         //初始化 SJA1000
```

```
    install_inter( );                       //装载中断
    inter_ocw=inportb(0x21);                //保存旧中断屏蔽字
    outportb(0x21,inter_ocw&0xf3);          //写中断屏蔽字，打开 IRQ9
    help( );                                //输出帮助信息
    do{
        in_key=getch( );                    //输入选择
        switch(in_key)
        {
        case '1':                           //输入'1'-CAN1 向 CAN2 发送数据
            printf("Can1 Send=");
            u_data=getch( );                //输入要发送的数据
            printf("%c\n",u_data);
            for(i=0;i<10;i++){send(1,Local_2,u_data);delay(50);}     //循环发送 10 次
            break;
        case '2':                           //输入'2'-CAN2 向 CAN1 发送数据
            printf("Can2 Send=");
            u_data=getch( );                //输入要发送的数据
            printf("%c\n",u_data);
            for(i=0;i<10;i++){send(2,Local_1,u_data);delay(50);}     //循环发送 10 次
            break;
        case '3':                           //输入'3'-复位 1 号 SJA1000
            printf("Reset Can1.\n");
            reset_can(1);
            break;
        case '4':                           //输入'4'-复位 2 号 CAN2
            printf("Reset Can2.\n");
            reset_can(2);
            break;
        case '5':                           //输入'5'-初始化 SJA1000
            printf("Initial Can.\n");
            initial_can( );
            break;
        case 'q':                           //输入'q'-结束程序
            resume_inter( );
            printf("Test over.\n");
            break;
        default:                            //输入其他值显示帮助信息
            help( );
        }
    }while(in_key!='q');
}

void help( )      //帮助信息
{
    printf("\n===============Test PC-104_CAN================\n");
```

```
	printf("'1' for can1 to can2 , '2' for can2 to can1\n");
	printf("'3' for reset can1, '4' for reset can2\n");
	printf("'5' for initial can.\n");
	printf("'q' for exit.\n");
	printf("===========================================\n");
}

void initial_can( )                    //初始化 SJA1000
{
	disable( );
	//initial can1
	SJA_Write(1,0,0x01);
	SJA_Write(1,4,Local_1);     //站地址
	SJA_Write(1,5,0x00);
	SJA_Write(1,6,0x47);        //波特率为 50kbit/s
	SJA_Write(1,7,0x2f);
	SJA_Write(1,8,0xda);        //推挽输出
	SJA_Write(1,0,0x72);
	//initial can2
	SJA_Write(2,0,0x01);
	SJA_Write(2,4,Local_2);     //站地址
	SJA_Write(2,5,0x00);
	SJA_Write(2,6,0x47);        //波特率为 50kbit/s
	SJA_Write(2,7,0x2f);
	SJA_Write(2,8,0xda);        //推挽输出
	SJA_Write(2,0,0x72);
	enable( );
}

void send(int num,unsigned char aim_adress,unsigned char out_data)
{
	unsigned char temp;
	temp=SJA_Read(num,2);
	if((temp&0x04)==0x04)                   //TBF 是否可写
	{                                       //是,则写入报文
		SJA_Write(num,10,aim_adress);   //写目标地址
		SJA_Write(num,11,1);            //写发送字节数
		SJA_Write(num,12,out_data);     //写发送数据
		SJA_Write(num,1,0x01);          //启动发送
	}
}

void SJA_Write(int num,unsigned char in_adress,unsigned char out_data)          //写 SJA1000
{
	outportb(Basic_Adress,in_adress);                   //写入片内寄存器地址
```

```
    if(num==1)outportb(Basic_Adress+2,out_data);        //向 1 号 SJA1000 写入数据
    if(num==2)outportb(Basic_Adress+3,out_data);        //向 2 号 SJA1000 写入数据
}

unsigned char SJA_Read(int num,unsigned char in_adress)//读 SJA1000
{
    unsigned char in_data;
    outportb(Basic_Adress,in_adress);                   //写入片内寄存器地址
    if(num==1)in_data=inportb(Basic_Adress+2);          //从 1 号 SJA1000 读出数据
    if(num==2)in_data=inportb(Basic_Adress+3);          //从 2 号 SJA1000 读出数据
    return(in_data);   //返回数据
}

void reset_can(int num)                                 //复位 SJA1000
{                                                       //D0 for can1;D1 for can2
    if(num==1)                                          //复位 1 号 SJA1000
    {
        outportb(Basic_Adress+1,0x02);
        delay(50);
        outportb(Basic_Adress+1,0x03);
    }
    if(num==2)                                          //复位 2 号 SJA1000
    {
        outportb(Basic_Adress+1,0x01);
        delay(50);
        outportb(Basic_Adress+1,0x03);
    }
}

void install_inter( )                                   //装载中断
{
    disable( );                                         //关中断
    old_can_inter=getvect(Int_num);                     //保存旧中断入口
    old_time_inter=getvect(Int_time);
    setvect(Int_num,can_inter);                         //设置新中断入口
    setvect(Int_time,time_inter);
    enable( );                                          //开中断
}

void interrupt can_inter( )                             //CAN 接收中断
{
    unsigned char temp;
    disable( );                                         //关中断
    //can1 interrupt
    temp=SJA_Read(1,3);
```

```
    if(temp&0x08)                                   //是否超载
    {
        SJA_Write(1,1,0x08);                        //是,则清除超载状态
    }
    if(temp&0x01)                                   //若有接收中断
    {
        canbuf[1]=SJA_Read(1,22);                   //则将数据写入缓冲区
        SJA_Write(1,1,0x0c);                        //释放 TBF
        putchar(canbuf[1]);                         //显示数据
        puts("=>Can1 Receive");
    }
    //can2 interrupt
    temp=SJA_Read(2,3);
    if(temp&0x08)                                   //是否超载
    {
        SJA_Write(2,1,0x08);                        //是,则清除超载状态
    }
    if(temp&0x01)                                   //若有接收中断
    {
        canbuf[2]=SJA_Read(2,22);                   //则将数据写入缓冲区
        SJA_Write(2,1,0x0c);                        //释放 TBF
        putchar(canbuf[2]);                         //显示数据
        puts("=>Can2 Receive");
    }
    outportb(0x20,0x20);                            //发中断结束命令
    enable( );                                      //开中断
}

void interrupt time_inter( )                        //定时检测 SJA1000 是否总线脱离
{
    unsigned char temp;
    disable( );                                     //关中断
    temp=SJA_Read(1,2);
    if((temp&0x80)==0x80)
    {
        SJA_Write(1,0,0x01);                        //若脱离,则将 SJA1000 复
        SJA_Write(1,0,0x72);                        //在进入正常工作方式
    }
    temp=SJA_Read(2,2);
    if((temp&0x80)==0x80)
    {
        SJA_Write(2,0,0x01);                        //若脱离,则将 SJA1000 复位
        SJA_Write(2,0,0x72);                        //在进入正常工作方式
    }
    enable( );                                      //开中断
```

```
}

void resume_inter( )                                        //恢复中断
{
      disable( );                                           //关中断
      setvect(Int_num,old_can_inter);                       //恢复旧中断入口
      setvect(Int_time,old_time_inter);
      outportb(0x21,inter_ocw);                             //恢复旧中断屏蔽字
      enable( );                                            //开中断
}
```

6.2 ISA 总线 CAN 接口卡设计

6.2.1 ISA 总线简介

ISA（Industrial Standard Architecture）总线是 IBM 公司 1984 年为推出 PC/AT 机而建立的系统总线标准，所以也叫 AT 总线，它是对 XT 总线的扩展。486、586、PII 微机都提供 ISA 总线标准。关于 ISA 总线详细资料可参考相关的手册，表 6-1 为 ISA 总线的信号中与 I/O 读写有关的信号。以下主要说明 ISA 总线的读写时序如图 6-11 所示。

表 6-1　ISA 总线的信号中与 I/O 读写有关的信号

符　　号	引　　脚	说　　明
BALE	B28	地址锁存允许信号，下降沿锁存地址（未用）
SA0～SA15	A31～A16	系统地址总线
SD0～SD7	A9～A2	系统数据总线
$\overline{IOR}$	B14	I/O 读信号
$\overline{IOW}$	B13	I/O 写信号
I/OCH RDY	A10	I/O 通道准备好，用以延长 I/O 读写周期（未用）
AEN	A11	DMA 地址允许信号，在 DMA 操作时有效

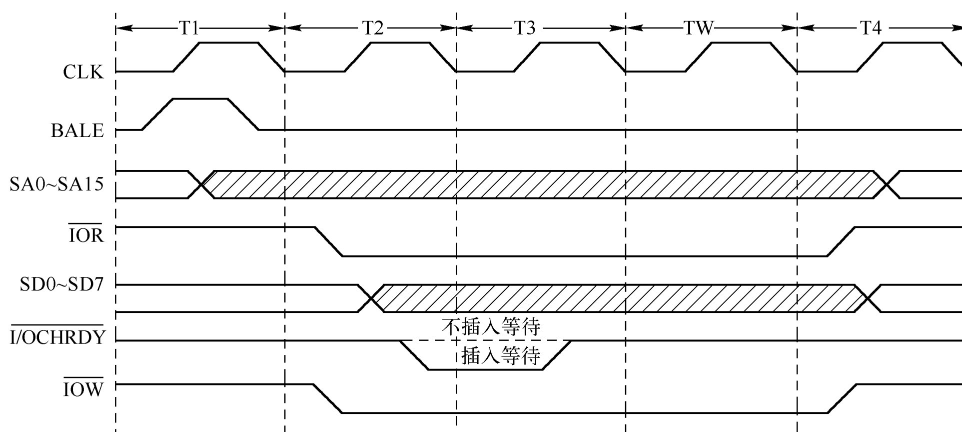


图 6-11　8bit I/O 读/写周期时序图

6.2.2 硬件电路设计说明

ISA 总线与 PC-104 总线在总线定义和读写时序等方面是完全相同的，二者的主要区别在于外部的硬件接口形式和电路板的尺寸要求上。因此，从电气接口设计和软件设计的角度来看，基于 ISA 总线和 PC-104 总线的 CAN 接口卡基本相同。

1．基地址设置

一般 PC 在进行 I/O 读写操作时只使用了 SA0～SA9 十根地址线作为寻址线，此卡使用了 SA2～SA9 八根地址线作为基地址（BASE_ADDRESS）译码信号，通过与卡上预设的地址相比较输出基地址有效信号 $\overline{\text{CS}}$，如图 6−12 所示，SA0、SA1 作为卡内地址参与译码，共使用了从基地址开始的连续 4 个 I/O 地址。

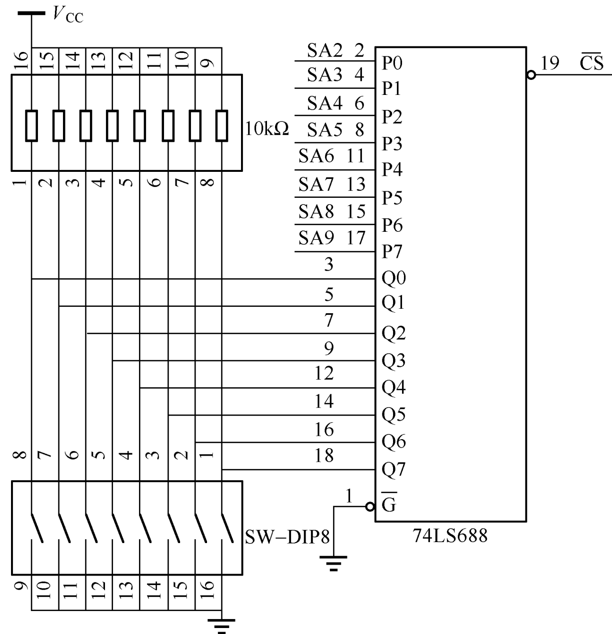


图 6−12　译码电路

2．I/O 读写接口

ISA 总线的地址和数据线是分离的，而 SJA1000 的地址和数据是复用的，所以无法直接相连。可以使用两个连续的 ISA 总线的 I/O 操作合成一个对 SJA1000 的 I/O 操作，即第一次 I/O 周期通过数据线向 SJA1000 写入片内寄存器地址，第二次 I/O 周期从 SJA1000 写入（读出）数据。

此卡是双 CAN 卡，有两片 SJA1000，编为 1 号、2 号，需要的读写信号有：

$\overline{\text{CS1}}$——1 号 SJA1000 片选信号；

$\overline{\text{CS2}}$——2 号 SJA1000 片选信号；

ALE——地址锁存信号；

$\overline{\text{WR}}$——读信号；

$\overline{\text{RD}}$——写信号。

同时要避免 PC 在进行 DMA 操作时可能发生的地址冲突。

信号的逻辑关系如图 6-13 所示：

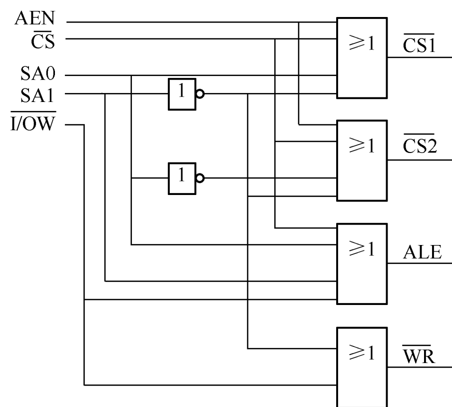


图 6-13　控制电路

RD 与*I/OR 直接相连。

要完成一次读/写操作，首先向（BASE_ADDRESS+0）端口写入片内寄存器地址，再从（BASE_ADDRESS+2）端口（1 号）读出/写入数据，或者从（BASE_ADDRESS+3）端口（2 号）读出/写入数据。进行 I/O 读写的时序图如图 6-14 所示。

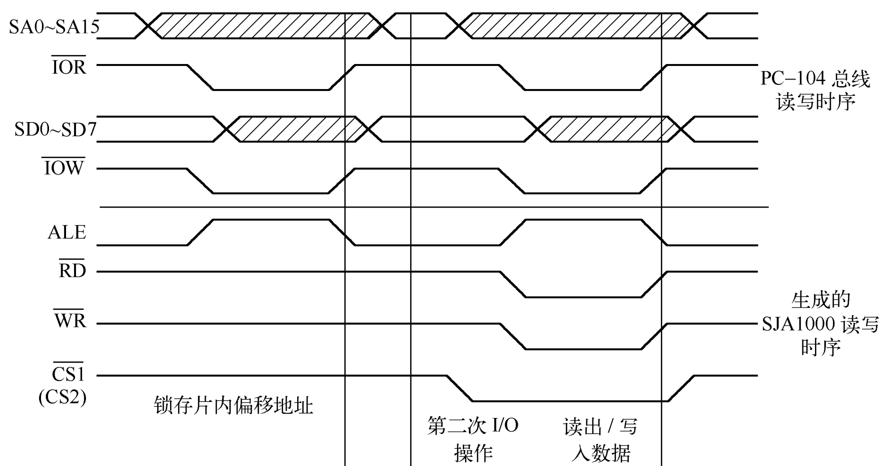


图 6-14　I/O 读写时序图

在 PC 进行 DMA 操作时 AEN 有效，使 CS1、CS2 无效。

3．中断接口

ISA 总线的中断输入信号 IRQx 为高电平有效，SJA1000 的中断输出信号为低电平有效。二者的接口需要一个非门连接，为了节省 PC 的资源，可以通过短接器短接两个 SJA1000 的中断输出，使其共用一个中断。同时考虑到保持与 PC 其他中断资源的线或关系，采用了开漏输出的非门 7406。中断接口如图 6-15 所示。

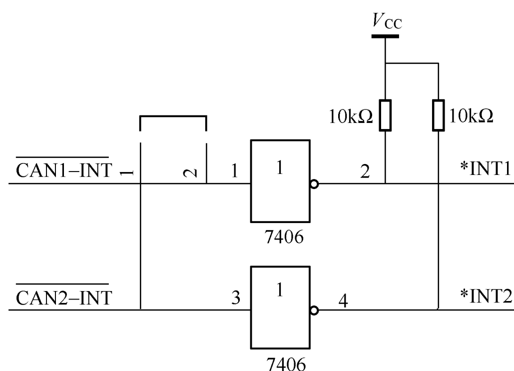


图 6-15　中断控制电路+

4．复位信号

SJA1000 的复位输入信号为 $\overline{\text{RESET1}}$（$\overline{\text{RESET2}}$），ISA 总线的上电复位输出信号为 *RESET，接入两个上升沿锁存的 D 触发器 7474，实现复位接口。SJA1000 在以下两种情况下复位：

① PC 启动上电时，两个 SJA1000 均复位。

② 对 1 号 SJA1000：向（BASE_ADDRESS+1）端口写入数据 0x02，延时一段时间，再写入 0x03；对 2 号 SJA1000：向（BASE_ADDRESS+1）端口写入数据 0x01，延时一段时间，再写入 0x03。复位电路如图 6-16 所示。

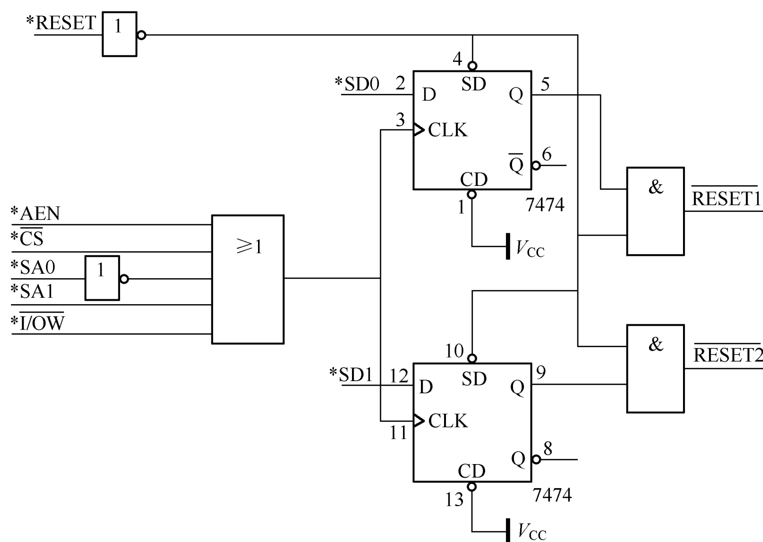


图 6-16　复位电路

通信卡上逻辑电路是用一片 GAL16V8 实现的。

5．CAN 网输出接口

主要是 SJA1000 与 82C250 的连接，为了保护 PC 加入了高速光耦 6N137，并采取了电源隔离措施。图 6-17 是 1 号 SJA1000 的连线图（2 号与此完全类似）：



图 6-17　CAN 接口电路

电源隔离采用了 5V/5V 的 DC/DC 电源隔离器，如图 6-18 所示。

图 6-18　电源隔离电路

CAN 总线输出接口采用了标准的 9 针串口接插件，可以通过跳线器使卡内的两个 CAN 节点互联，选择接入终端电阻（120Ω）。CAN 总线输出接口如图 6-19 所示。

图 6-19　CAN 总线输出接口电路

基于 ISA 总线的双 CAN 接口卡总电路原理图如图 6-20 所示。

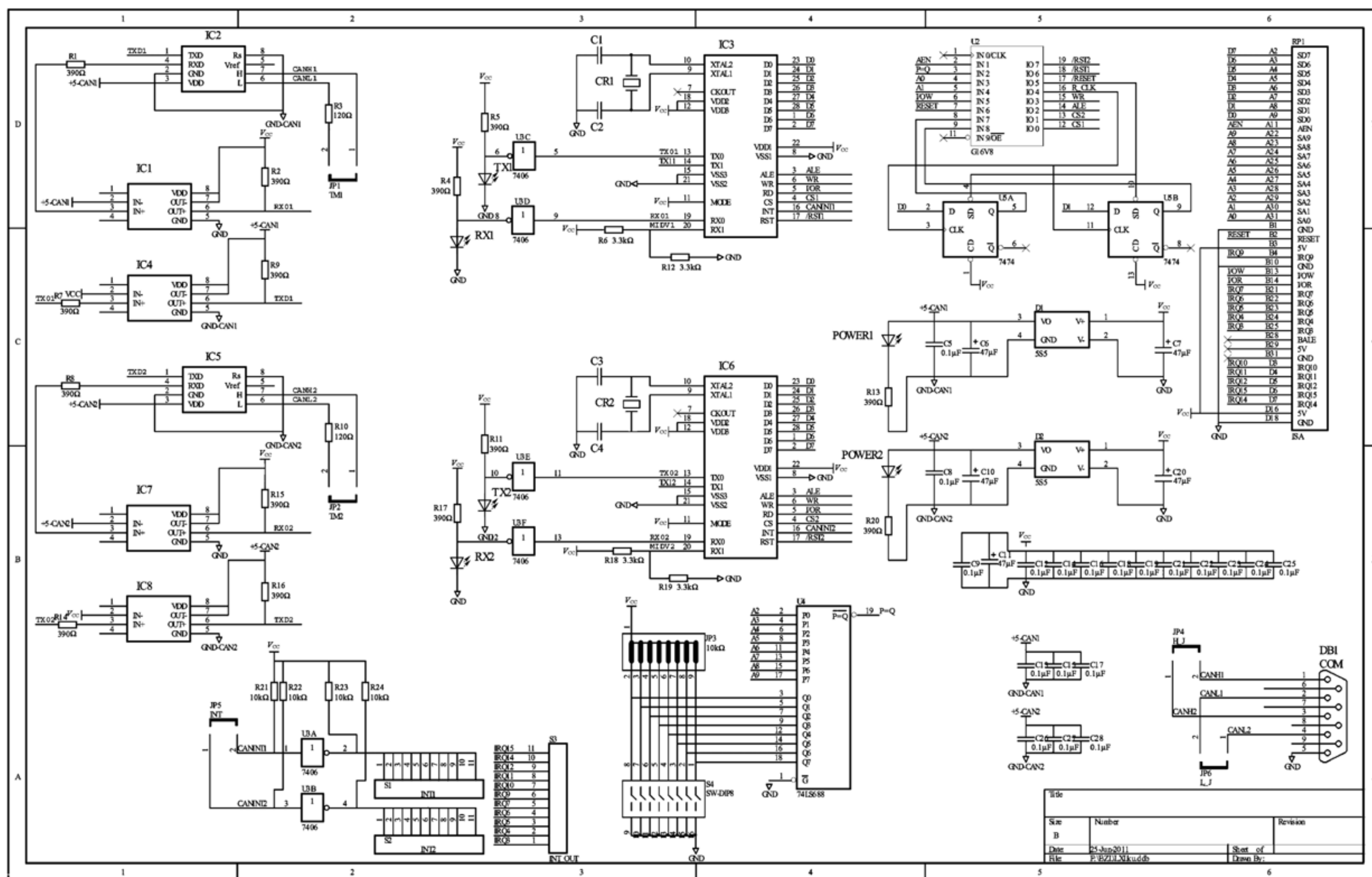


图 6-20　ISA 总线接口卡电路原理图

6.2.3 ISA 接口卡软件设计

基于 ISA 总线的 CAN 接口卡通信程序与 PC-104 接口卡程序基本相同，在此只给出完整的测试程序，其中子程序的功能说明参照 6.1.3 节中的相关说明。

```
/*****************************************************************************
 基于 ISA 总线的 CAN 网通信卡的配套测试程序
 编程语言：C 语言
 运行环境：DOS
 测试功能：测试同一块卡上的两个 SJA1000 之间的通信。
 注意事项：测试前将通信板的基地址设置为 0x220，中断设置为共用 IRQ9，短接 TM1、TM2、
           H_J、L_J。
 设置完成后将通信卡插入 PC 的 ISA 插槽，运行测试程序，根据软件的提示操作即可。
*****************************************************************************/
#include "stdio.h"
#include "dos.h"
#define Basic_Adress 0x220          //基地址
#define Int_num 0x0a                //中断向量号 IRQ9 0x0a
#define Int_time 0x1c               //定时中断向量号
#define   Local_1    1              //1 号 CAN 的本站地址
#define   Local_2    2              //2 号 CAN 的本站地址

void interrupt can_inter( );        //CAN 接收中断
void interrupt time_inter( );       //定时中断
void interrupt (*old_can_inter)( ); //保存旧中断入口
void interrupt (*old_time_inter)( );
void install_inter( );              //装载中断
void resume_inter( );               //恢复中断
void initial_can( );                //初始化 SJA1000

/*写 SJA1000，参数：num-选择 SJA1000(1-1 号 SJA1000，2-2 号 SJA1000);
                  in_adress-SJA1000 内部寄存器地址
                  out_data-要写入的数据*/
void SJA_Write(int num,unsigned char in_adress,unsigned char out_data);

/*读 SJA1000，参数：num-选择 SJA1000(1-1 号 SJA1000，2-2 号 SJA1000);
                  in_adress-SJA1000 内部寄存器地址
                  返回值-读出的数据*/
unsigned char SJA_Read(int num,unsigned char in_adress);

/*发送数据，参数：  num-选择 SJA1000(1-1 号 SJA1000，2-2 号 SJA1000);
                  aim_adress-目标地址
                  out_data-要发送的数据*/
void send(int num,unsigned char aim_adress,unsigned char out_data);

//复位 SJA1000，参数：num-选择 SJA1000(1-1 号 SJA1000，2-2 号 SJA1000)
void reset_can(int num);
```

```
void help( );                          //帮助信息

unsigned char canbuf[5],inter_ocw;     //全局变量：canbuf-输入缓存；inter_ocw-中断屏蔽字

main( )
{
unsigned char u_data;
char in_key;
unsigned long i;
initial_can( );                        //初始化 SJA1000
install_inter( );                      //装载中断
inter_ocw=inportb(0x21);               //保存旧中断屏蔽字
outportb(0x21,inter_ocw&0xf3);         //写中断屏蔽字，打开 IRQ3、IRQ9
help( );                               //输出帮助信息
do{
    in_key=getch( );                   //输入选择
    switch(in_key)
    {
    case '1':                          //输入'1' –CAN1 向 CAN2 发送数据
        printf("Can1 Send=");
        u_data=getch( );               //输入要发送的数据
        printf("%c\n",u_data);
        for(i=0;i<10;i++){send(1,Local_2,u_data);delay(50);}     //循环发送 10 次
        break;
    case '2':                          //输入'2' –CAN2 向 CAN1 发送数据
        printf("Can2 Send=");
        u_data=getch( );               //输入要发送的数据
        printf("%c\n",u_data);
        for(i=0;i<10;i++){send(2,Local_1,u_data);delay(50);}     //循环发送 10 次
        break;
    case '3':                          //输入'3' –复位 1 号 SJA1000
        printf("Reset Can1.\n");
        reset_can(1);
        break;
    case '4':                          //输入'4' –复位 2 号 CAN2
        printf("Reset Can2.\n");
        reset_can(2);
        break;
    case '5':                          //输入'5' –初始化 SJA1000
        printf("Initial Can.\n");
        initial_can( );
        break;
    case 'q':                          //输入'q' –结束程序
        resume_inter( );
        printf("Test over.\n");
```

```
            break;
        default:                            //输入其他值显示帮助信息
            help( );
        }
}while(in_key!='q');
}

void help( )                                //帮助信息
{
printf("\n===============Test ISA_CAN================\n");
printf("'1' for can1 to can2 , '2' for can2 to can1\n");
printf("'3' for reset can1      , '4' for reset can2\n");
printf("'5' for initial can.\n");
printf("'q' for exit.\n");
printf("============================================\n");
}

void initial_can( )                         //初始化 SJA1000
{
disable( );
//initial can1
SJA_Write(1,0,0x01);
SJA_Write(1,4,Local_1);                     //站地址
SJA_Write(1,5,0x00);
SJA_Write(1,6,0x47);                        //波特率为 50kbit/s
SJA_Write(1,7,0x2f);
SJA_Write(1,8,0xda);                        //推挽输出
SJA_Write(1,0,0x72);
//initial can2
SJA_Write(2,0,0x01);
SJA_Write(2,4,Local_2);                     //站地址
SJA_Write(2,5,0x00);
SJA_Write(2,6,0x47);                        //波特率为 50kbit/s
SJA_Write(2,7,0x2f);
SJA_Write(2,8,0xda);                        //推挽输出
SJA_Write(2,0,0x72);
enable( );
}

void send(int num,unsigned char aim_adress,unsigned char out_data)
{
unsigned char temp;
temp=SJA_Read(num,2);
if((temp&0x04)==0x04)                       //TBF 是否可写
{                                           //是,则写入报文
SJA_Write(num,10,aim_adress);               //写目标地址
```

```
  SJA_Write(num,11,1);                                //写发送字节数
  SJA_Write(num,12,out_data);                         //写发送数据
  SJA_Write(num,1,0x01);                              //启动发送
 }
}

void SJA_Write(int num,unsigned char in_adress,unsigned char out_data)           //写 SJA1000
{
outportb(Basic_Adress,in_adress);                     //写入片内寄存器地址
if(num==1)outportb(Basic_Adress+2,out_data);          //向 1 号 SJA1000 写入数据
if(num==2)outportb(Basic_Adress+3,out_data);          //向 2 号 SJA1000 写入数据
}

unsigned char SJA_Read(int num,unsigned char in_adress)        //读 SJA1000
{
unsigned char in_data;
outportb(Basic_Adress,in_adress);                     //写入片内寄存器地址
if(num==1)in_data=inportb(Basic_Adress+2);            //从 1 号 SJA1000 读出数据
if(num==2)in_data=inportb(Basic_Adress+3);            //从 2 号 SJA1000 读出数据
return(in_data);                                      //返回数据
}

void reset_can(int num)                               //复位 SJA1000
{//D0 for can1;D1 for can2
if(num==1)                                                 //复位 1 号 SJA1000
{
     outportb(Basic_Adress+1,0x02);
     delay(50);
     outportb(Basic_Adress+1,0x03);
}
if(num==2)                                                 //复位 2 号 SJA1000
{
     outportb(Basic_Adress+1,0x01);
     delay(50);
     outportb(Basic_Adress+1,0x03);
 }
}

void install_inter( )                                 //装载中断
{
     disable( );                                      //关中断
     old_can_inter=getvect(Int_num);                  //保存旧中断入口
     old_time_inter=getvect(Int_time);
     setvect(Int_num,can_inter);                      //设置新中断入口
     setvect(Int_time,time_inter);
     enable( );   //开中断
```

```
}

void interrupt can_inter( )                         //CAN 接收中断
{
        unsigned char temp;
        disable( );                                 //关中断
        //can1 interrupt
        temp=SJA_Read(1,3);
        if(temp&0x08)                               //是否超载
        {
                SJA_Write(1,1,0x08);                //是,则清除超载状态
        }
        if(temp&0x01)                               //若有接收中断
        {
                canbuf[1]=SJA_Read(1,22);           //则将数据写入缓冲区
                SJA_Write(1,1,0x0c);                //释放 TBF
                putchar(canbuf[1]);                 //显示数据
                puts("=>Can1 Receive");
        }
        //can2 interrupt
        temp=SJA_Read(2,3);
        if(temp&0x08)                               //是否超载
        {
                SJA_Write(2,1,0x08);                //是,则清除超载状态
        }
        if(temp&0x01)                               //若有接收中断
        {
                canbuf[2]=SJA_Read(2,22);           //则将数据写入缓冲区
                SJA_Write(2,1,0x0c);                //释放 TBF
                putchar(canbuf[2]);                 //显示数据
                puts("=>Can2 Receive");
        }
        outportb(0x20,0x20);                        //发中断结束命令
        enable( );                                  //开中断
}

void interrupt time_inter( )                        //定时检测 SJA1000 是否总线脱离
{
        unsigned char temp;
        disable( );                                 //关中断
        temp=SJA_Read(1,2);
        if((temp&0x80) ==0x80)
        {
                SJA_Write(1,0,0x01);                //若脱离,则将 SJA1000 复位
                SJA_Write(1,0,0x72);                //在进入正常工作方式
        }
```

```
        temp=SJA_Read(2,2);
        if((temp&0x80)==0x80)
        {
              SJA_Write(2,0,0x01);              //若脱离,则将 SJA1000 复位
              SJA_Write(2,0,0x72);              //在进入正常工作方式
        }
        enable( );          //开中断
}

void resume_inter( )                            //恢复中断
{
        disable( );                             //关中断
        setvect(Int_num,old_can_inter);         //恢复旧中断入口
        setvect(Int_time,old_time_inter);
        outportb(0x21,inter_ocw);               //恢复旧中断屏蔽字
        enable( );                              //开中断
}
```

6.3 PCI 总线 CAN 接口卡设计

6.3.1 PCI 总线简介

外部设备互联总线（Peripheral Component Interconnect，PCI）是一种具有多路地址线和数据线的高性能的 32/64bit 总线。它在高度集成的外围控制器件、外围插件板和处理器/存储器之间作为互连机构应用。PCI 总线主要有以下几个特点：

1）数据传输速度高，PCI 总线时钟频率为 33MHz/66MHz，峰值传输速度为 132MB/s，528MB/s。

2）支持即插即用（Plug-and-Play，PnP），方便用户的使用。

3）多总线主控方式，适应于点对点的大量数据传输。

4）线性突发传输（Burst），数据帧传输模式。

5）兼容性强，通过各种桥接设备可以方便地转接到其他总线。

PCI 总线目前已成为 PC 的主流配置，逐步取代了原有的 ISA 总线。所以开发设计基于 PCI 总线的通信卡更具有发展前景。

在一个 PCI 应用系统中，获得总线控制权的设备称为“主设备”，而被主设备选中进行通信的设备称为“从设备”或“目标设备”。CAN 适配卡属于“从设备”，在通信过程中 PC 始终保持总线的控制权，在 PC 的控制下完成数据的双向传输，从而实现 PC 和 CAN 网络的通信。

选用 SJA1000 作为 CAN 控制器，要解决的问题就是如何通过 PCI 总线实现对 SJA1000 内部寄存器的读写，以及中断信号的响应。PCI 总线的特点在给用户的使用带来方便的同时，也增加了用户的开发难度。主要是为了实现即插即用等功能，总线在时序上比较复杂，需要一个协同的配置过程。不能用类似 ISA 总线开发那样简单的逻辑电路实现。目前主要的开发手段有两种，一种是利用 CPLD 或 FPGA 技术，在内部实现总线的规范，这种方法的优点是

开发灵活，集成度高，但是开发的难度大，需要对 PCI 总线规范有很深入的了解，设计周期长。另一种方法是采用 PCI 总线专用接口芯片，这类芯片在内部实现了 PCI 总线的规范，提供给外部的是用户可配置的本地总线接口以及中断响应，开发者进行简单的设置就可以实现和 PCI 总线的接口。

根据开发需要，选取了 PLX 公司的 PCI9052 专用接口芯片，PCI9052 是一种性能优秀而且价格低廉的 PCI 总线目标接口芯片。根据用户的配置，PCI9052 将 PCI 总线信号转化成为各种不同的本地总线，使用户的设备很容易挂接到 PCI 总线中去，绕开了 PCI 总线规范带来的开发难题，大大简化了开发难度。

6.3.2 硬件电路设计说明

如前所述，PCI9052 的作用就是实现 PCI 总线和本地总线的连接，对于开发者来说，这种连接是透明的，不用考虑 PCI 总线的复杂规范，只需要根据 PCI 总线的信号定义连接到 PCI9052 相应的管脚，同时注意与 PCI 总线相关的布线规范就可以了。更多关心的是本地总线的配置，PCI9052 提供的本地总线从信号时序的角度可分为数据、地址复用模式，数据、地址分用模式和 ISA 模式。根据 SJA1000 的连接特点，选择了数据、地址复用模式。总线连接如图 6-21 所示。

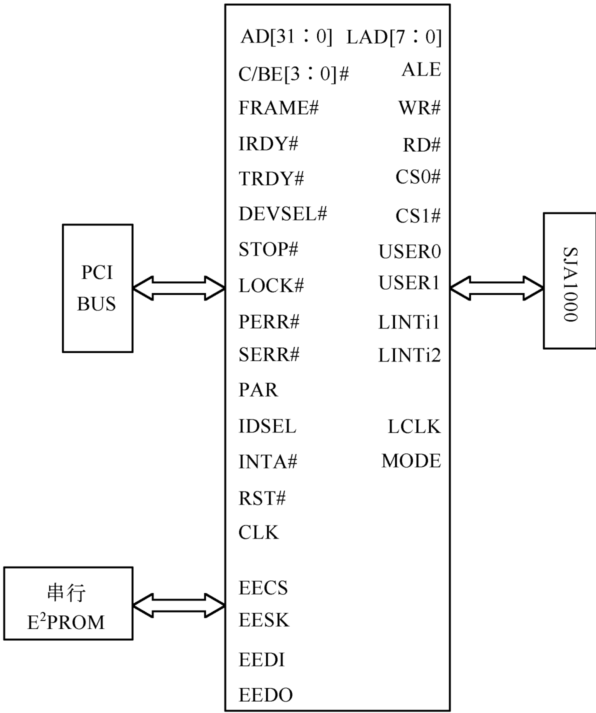


图 6-21　PCI9052 信号连接示意图

图 6-21 主要体现了信号的连接关系，具体的管脚和信号定义参照 PCI9052 的数据手册和 PCI 总线的相关规范。

从图中可以看出，左侧为 PCI9052 和 PCI 总线以及配置所用的 E^2PROM 的接口，在电路

连接上主要根据 PCI 总线的规范连接对应的信号线就可以了。E^2PROM 中保存的是用户的配置信息，在系统上电时 PCI9052 会根据里面的配置信息设置本地总线的工作方式，以及向主机申请本地所需的资源。

右侧为本地总线以及一些必要的设置管脚，LCLK 为本地总线的工作时钟，可根据需要提供不同频率的总线速度，在此设计中采用了 10MHz 的本地总线时钟。MODE 为本地总线模式选择，常接高使本地总线工作在数据、地址复用模式。

考虑设计 PCI 接口的双 CAN 接口卡，本地有两片 SJA1000。LAD[7∶0]为数据、地址复用总线，连接 SJA1000 的 AD[7∶0]，ALE 为地址锁存信号，WR＃和 RD＃为写信号和读信号，直接连接 SJA1000 的 ALE、WR#和 RD#。CS0#和 CS1#为两个独立的片选信号，分别连接两个 SJA1000 的 CS#。LINTi1 和 LINTi2 为本地中断输入，连接两个 SJA1000 的 INT＃输出。虽然有两个独立的本地中断申请，但 PCI9052 向主机申请中断时只产生一个中断申请，在响应中断时可以根据 PCI9052 相关寄存器的内容或 SJA1000 的状态判断是哪一个发出的中断请求。USER0 和 USER1 两路是可控的 I/O，分别连接两个 SJA1000 的复位线 RST#，实现可控的硬件复位。

在一定的配置条件下，PCI9052 的本地总线可以实现 SJA1000 所需的读写时序，从而简化了电路的设计。PCI9052 的配置工作除了对一些必要的设置信号线进行置高、拉低处理外，主要集中在 E^2PROM 内容的设置上。

在系统上电后 PCI9052 会根据 E^2PROM 中的内容设置本地总线，同时向主机申请本地所需的资源。所以正确的设置是电路工作的基础。

串行 E^2PROM 中存储了 PCI9052 重要的配置信息，主要有设备号 DID、制造商 VID、子设备号 SDID，子制造商 SVID、中断号、设备类型号、局部空间基地址、局部空间大小及映射类型、局部空间描述、片选响应、中断控制和状态以及局部响应控制 CNTRL 等信息。串行 E^2PROM 的内容直接关系到 PCI9052 能否正确的工作，若配置不正确，即使硬件设计没有任何错误，PCI9052 也很难正确地工作。对于本设计，串行 E^2PROM 的内容需要实现如下功能。

根据电路连接关系，需要将 SJA1000 的内部寄存器空间映射到 PC 的 I/O 空间，为两个连续的 128B I/O 空间提供两个片选信号 CS0#和 CS1#，工作方式为数据、地址复用形式。实现 USER0、1 的可控复位，以及本地两路中断请求。按照 PCI9052 手册上的说明，向 E^2PROM 中写入配置值，既可以向主机申请获得两个 128B 的连续 I/O 空间，一个中断资源，又可以通过 PCI9052 将此空间映射到本地的数据总线，同时将本地的中断请求转化为设备对主机的请求。当配置结束后，用户访问主机分配给 CAN 通信卡的 I/O 空间就可以完成对 SJA1000 的读写，响应通信卡的中断请求，或者通过改写特殊的寄存器来完成 SJA1000 的硬件复位。从这点来看，可以认为 PCI9052 是一个透明的桥接设备，为用户的软件编写也带来了很大的方便。

CAN 网络输出接口的硬件设计与 6.1.2 节基本相同，采取了光耦隔离的保护模式。

基于 PCI 总线的双 CAN 接口卡总电路原理图如图 6−22 所示。

6.3.3　PCI 接口卡软件设计

软件是使用 C 语言编写的，主要完成两个方面的工作。首先是获取主机分配给通信卡资

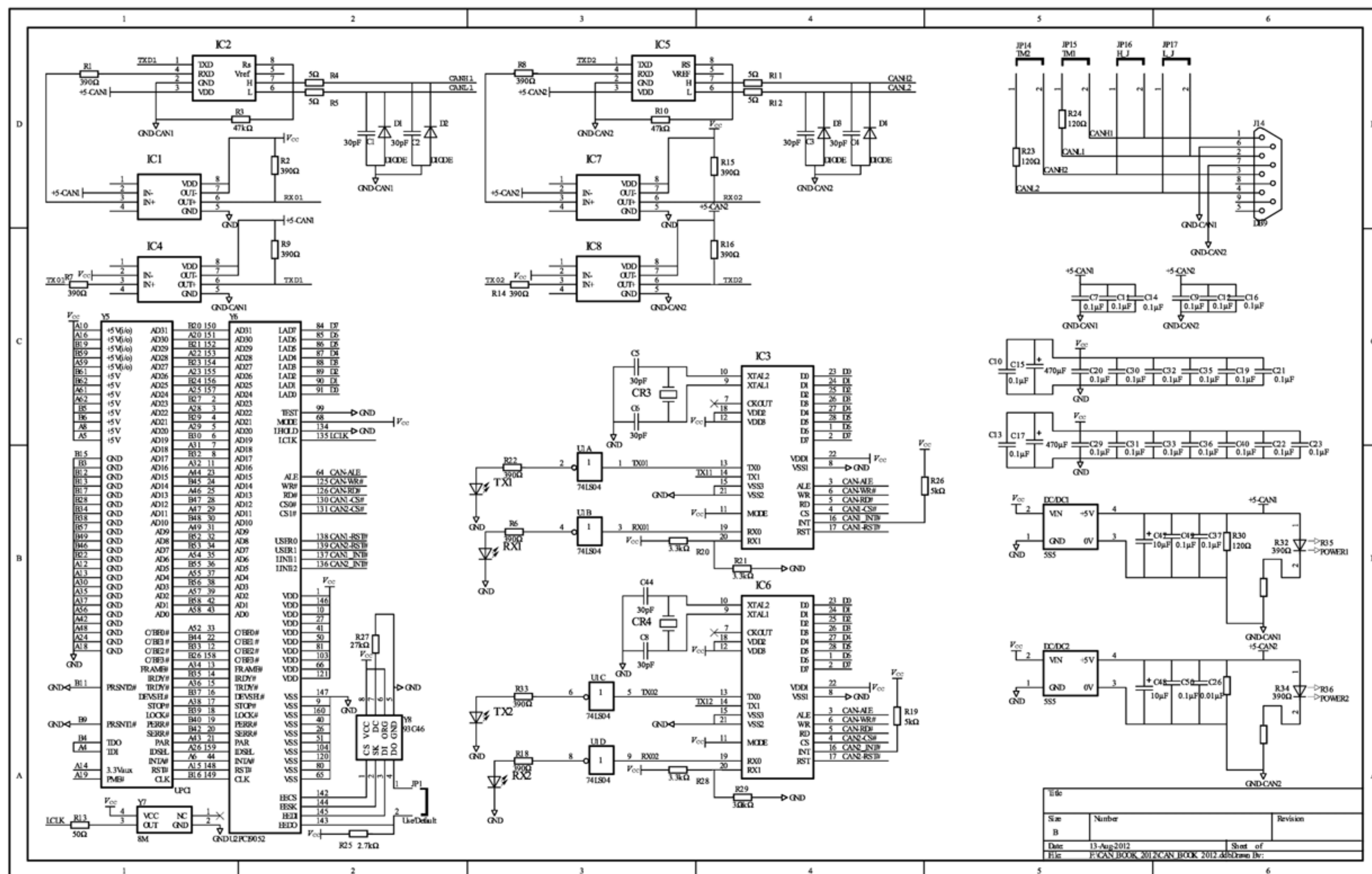


图 6-22　PCI 总线接口卡电路原理图

源的信息，主要是两个 I/O 空间的基地址，中断向量。其次是 CAN 总线通信程序，包括 SJA1000 的初始化，发送报文，接收报文以及对一般错误的处理。

```
struct PCI_CFG                          //定义结构体
{
        unsigned char flag;             //查询返回结果，成功返回 0，否则返回 1
        unsigned int sja1_bas;          //1 号 SJA1000 对应的基地址
        unsigned int sja2_bas;          //2 号 SJA1000 对应的基地址
        unsigned int cntr_add;          //控制复位寄存器地址
        unsigned int setinter;          //中断控制寄存器地址
        unsigned int int_num;           //中断向量号
};

#include <dos.h>                        //获取分配信息
#define DID 0x9050
#define VID 0x10b5
unsigned int extern rdl(unsigned char bn,unsigned char nf,unsigned char offset);

struct PCI_CFG sjaopen(void)
{
        unsigned char bus_no,fix_inf;
        unsigned int bar3_bas,bar2_bas,bar1_bas;  //基地址
        union REGS regs;

        struct PCI_CFG pcicfg;

        regs.h.ah=0xb1;                                 //检测 PCI BOIS 是否存在
        regs.h.al=0x01;
        int86(0x1a,&regs,&regs);                        //如果存在 PCI BOIS 返回 0
        if((!(regs.h.al&0x01))||(regs.h.ah))
        {pcicfg.flag=0xff;return(pcicfg);}
        regs.h.ah=0xb1;                                 //检测设备的存在性,并获取总线号和设备信息
        regs.h.al=0x02;
        regs.x.cx=DID;
        regs.x.dx=VID;
        regs.x.si=0;
        int86(0x1a,&regs,&regs);
        if(regs.h.ah)
        {
                pcicfg.flag=0xff;return(pcicfg);
        }                   //成功获取设备返回 0
        bus_no=regs.h.bh;
        fix_inf=regs.h.bl;
        bar1_bas=rdl(bus_no,fix_inf,0x14);              //BAR1 基地址
        bar1_bas=bar1_bas&0xfffe;
```

```
        bar2_bas=rdl(bus_no,fix_inf,0x18);                  //BAR2 基地址
        bar2_bas=bar2_bas&0xfffe;
        pcicfg.sja1_bas=bar2_bas;
        bar3_bas=rdl(bus_no,fix_inf,0x1c);                  //BAR3 基地址
        bar3_bas=bar3_bas&0xfffe;
        pcicfg.sja2_bas=bar3_bas;
        pcicfg.cntr_add=bar1_bas+0x50;
        pcicfg.setinter=bar1_bas+0x4c;
        pcicfg.int_num=rdl(bus_no,fix_inf,0x3c)&0x00ff;
        outport(pcicfg.cntr_add,0x4db6);                     //取消复位，使 SJA1000 进入工作状态
        pcicfg.flag=0x00;
        return(pcicfg);                                     //返回所获得的信息
}
```

在程序开头调用函数 sjaopen( )，就可以获得主机分配给通信卡的资源，在这个基础上的读写等操作和对一般的 I/O 设备的操作就完全一样了。

```
unsigned char sja_read(int num,unsigned char in_adress)    //读 SJA1000
{
        unsigned char in_data;
        if(in_adress>=128)
        printf("\nOut of Range!\n");return(0x00);           //检测是否超界
        //从 1 号 SJA1000 读出数据
        if(num==1)
                in_data=inportb(sja_inf.sja1_bas+in_adress);
        //从 2 号 SJA1000 读出数据
        if(num==2)
                in_data=inportb(sja_inf.sja2_bas+in_adress);
        return(in_data);                                    //返回数据
}
void sja_write(int num,unsigned char in_adress,unsigned char out_data) //写 SJA1000
{
        if(in_adress>=128)
        {
                printf("\nOut of Range!\n");
                return;
        }                                                   //检测是否超界
        //向 1 号 SJA1000 写入数据
        if(num==1)
                outportb(sja_inf.sja1_bas+in_adress,out_data);
        //向 2 号 SJA1000 写入数据
        if(num==2)
                outportb(sja_inf.sja2_bas+in_adress,out_data);
}

void reset_sja(int num)                                     //硬件复位 SJA1000
{
```

```
        if(num= =1)                                  //复位 1 号 SJA1000
        {
        outport(sja_inf.cntr_add,0x4db2);
            delay(50);
            outport(sja_inf.cntr_add,0x4db6);
        }
         if(num= =2)                                 //复位 2 号 SJA1000
        {
            outport(sja_inf.cntr_add,0x4d96);
            delay(50);
            outport(sja_inf.cntr_add,0x4db6);
        }
    }
    //中断的使能与禁止
    void enable_inter( )                             //容许 SJA1000 中断
    {
        outport(sja_inf.setinter,0x0049);
    }

    void disable_inter( )                            //禁止 SJA1000 中断
    {
        outport(sja_inf.setinter,0x0000);
    }
```

有两个中断需要处理，一个是 SJA1000 接收中断；另一个是为了保证系统的可靠性需要定时检测 SJA1000 的状态，并做出相应的处理。所以中断程序包括对外中断的处理和定时产生一个软中断。分为中断的安装和退出时中断向量的恢复。

```
    int Int_vector[16] ={0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f, 0x70, 0x0a, 0x72, 0x73, 0x74,
0x75, 0x76, 0x77};
    int use_int;                                     //中断向量号
    unsigned char inter_ocw,inter_ocw2;              //中断屏蔽字

    void install_inter( )                            //装载中断
    {
        disable( );                                  //关中断
        use_int=Int_vector[sja_inf.int_num];
        old_time_inter=getvect(Int_time);            //保存旧中断入口
        old_res_inter=getvect(use_int);
        setvect(Int_time,time_inter);                //设置新中断入口
        setvect(use_int,res_inter);
        enable( );                                   //开中断
    }
    void resume_inter( )                             //恢复中断
    {
        disable( );                                  //关中断
        setvect(use_int,old_res_inter);              //恢复旧中断入口
```

```c
        setvect(Int_time,old_time_inter);
        outportb(0x21,inter_ocw);                   //恢复旧中断屏蔽字
        outportb(0xa1,inter_ocw2);
        enable( );                                  //开中断
}
```

当实现了对 SJA1000 的读写和中断的设置后，对于 SJA1000 的初始化、报文发送、报文接收以及错误处理等功能的实现与 6.1.3 节中相关内容基本相同。 基于 PCI 总线的双 CAN 接口卡通信测试源程序如下。

```c
/*****************************************************************************
基于 PCI 总线的 CAN 网通信卡的配套测试程序-主程序
编程语言：C 语言
运行环境：DOS
功能：测试 SJA1000 的初始化以及通信功能。
注意事项：
        在关机的状态下将通信卡插入 PC 的 PCI 插槽，运行设备检测程序 sjaopen( )[包含在 can.lib 库中]，确定是否安装正确；
        接收采用中断方式，在中断程序 res_inter( )中处理接收到的数据,并保存在输入缓存中；
        定时中断程序 time_inter( )以每秒 18.2 次的频率产生中断调用，在 time_inter( )中编写查询 SJA1000 状态的程序(可以省略)。
        主程序处理用户输入、输出以及显示等。
        通信卡上有两片 SJA1000，分别为 1 号和 2 号。
*****************************************************************************/

#include <dos.h>
#include <stdio.h>
#include <process.h>
#include "can.h"

//全局变量：
unsigned char res1buf[MAXnum],res2buf[MAXnum];      //输入缓存
unsigned char inter_ocw,inter_ocw2;                 //中断屏蔽字
struct PCI_CFG sja_inf;                             //设备检测返回值
int
Int_vector[16]={0x08,0x09,0x0a,0x0b,0x0c,0x0d,0x0e,0x0f,0x70,0x0a,0x72,0x73,0x74,0x75,0x76,0x77};
int use_int; //中断向量号
main( )
{
        char in_key;
        unsigned char u_data;
        unsigned long count;

        printf("\n*******************************************");
        printf("\n*        PCI_CAN_NET TEST Demo Program        *");
        printf("\n*          This ver3.0 use interrupt          *");
        printf("\n*******************************************\n\n");
```

```
	sja_inf=sjaopen( );	//检测设备，并获得 SJA1000 的基地址、配置地址、中断号
	if(sja_inf.flag){printf("\nWrong!Please check your hardware.\n");return;}	//设备出错，返回

	initial_sja( );				//初始化 SJA1000
	install_inter( );			//装载中断
	inter_ocw=inportb(0x21);		//保存旧中断屏蔽字
	inter_ocw2=inportb(0xa1);
	outportb(0x21,inter_ocw&0x00);		//写中断屏蔽字，打开中断
	outportb(0xa1,inter_ocw2&0x00);
	enable_inter( );			//容许 SJA1000 中断

//发送/接收程序
/*注意事项：
	发送一方：发送后延时等待一段时间（delay(50)）再发下一个数据
	接收一方：两个 SJA1000 共用一个中断，在程序中分别读取两个 SJA1000 的状态以确定是哪
片接收到了数据，并将接收到的数据存入缓存区.
	定时中断用于检测 SJA1000 的状态，如：是否脱离总线等,以决定是否复位（可省略）。
	在主程序里处理输入输出以及显示等。
*/

	help( );				//输出帮助信息
	do{
		in_key=getch( );		//输入选择
		switch(in_key)
		{
		case '1':			//输入'1'－CAN1 向 CAN2 发送数据
			printf("Can1 Send=");
			u_data=getch( );	//输入要发送的数据
			printf("%c\n",u_data);
			for(count=0;count<10;count++)
			{send(1,Local_2,u_data);delay(500);}	//循环发送 10 次
			break;
		case '2':			//输入'2'－CAN2 向 CAN1 发送数据
			printf("Can2 Send=");
			u_data=getch( );	//输入要发送的数据
			printf("%c\n",u_data);
			for(count=0;count<10;count++)
			{send(2,Local_1,u_data);delay(500);}	//循环发送 10 次
			break;
		case '3':			//输入'3'－复位 1 号 SJA1000
			printf("Reset SJA1000(1).\n");
			reset_sja(1);
			break;
		case '4':			//输入'4'－复位 2 号 SJA1000
```

```
            printf("Reset SJA1000(2).\n");
reset_sja(2);
                break;
            case '5':                                    //输入'5'－初始化 SJA1000
                printf("Initial SJA1000.\n");
                initial_sja( );
                break;
            case '6':                                    //输入'6'－容许 SJA1000 中断
                printf("Enable Interrupt.\n");
                enable_inter( );
                break;
            case '7':                                    //输入'7'－禁止 SJA1000 中断
                printf("Disable Interrupt.\n");
                disable_inter( );
                break;
            case 'q':                                    //输入'q'－结束程序
                break;
            default:                                     //输入其他值显示帮助信息
                help( );
            }
    }while(in_key!='q');
    disable_inter( );                                    //禁止 SJA1000 中断
    resume_inter( );                                     //恢复中断
    printf("\nTest over.\n");
}

void initial_sja( )                                      //初始化 SJA1000
{
    disable( );

    //initial can1
    sja_write(1,0,0x01);
    sja_write(1,4,Local_1);                              //站地址
    sja_write(1,5,0x00);
    sja_write(1,6,0x47);                                 //波特率为 50kbit/s
    sja_write(1,7,0x2f);
    sja_write(1,8,0xda);                                 //推挽输出
    sja_write(1,0,0x72);
    //initial can2
    sja_write(2,0,0x01);
    sja_write(2,4,Local_2);                              //站地址
    sja_write(2,5,0x00);
    sja_write(2,6,0x47);                                 //波特率为 50kbit/s
    sja_write(2,7,0x2f);
    sja_write(2,8,0xda);                                 //推挽输出
```

```
        sja_write(2,0,0x72);

enable( );
}

void sja_write(int num,unsigned char in_adress,unsigned char out_data)        //写 SJA1000
{
        if(in_adress>=128){printf("\nOut of Range!\n");return;}            //检测是否超界
        if(num==1)outportb(sja_inf.sja1_bas+in_adress,out_data);        //向 1 号 SJA1000 写入数据
        if(num==2)outportb(sja_inf.sja2_bas+in_adress,out_data);        //向 2 号 SJA1000 写入数据
}

unsigned char sja_read(int num,unsigned char in_adress)                    //读 SJA1000
{
        unsigned char in_data;
        if(in_adress>=128){printf("\nOut of Range!\n");return(0x00);}    //检测是否超界
        if(num==1)in_data=inportb(sja_inf.sja1_bas+in_adress);            //从 1 号 SJA1000 读出数据
        if(num==2)in_data=inportb(sja_inf.sja2_bas+in_adress);            //从 2 号 SJA1000 读出数据
        return(in_data);            //返回数据
}

void install_inter( )                                                      //装载中断
{
        disable( );                                                        //关中断
        use_int=Int_vector[sja_inf.int_num];
        old_time_inter=getvect(Int_time);                                  //保存旧中断入口
        old_res_inter=getvect(use_int);
        setvect(Int_time,time_inter);                                      //设置新中断入口
        setvect(use_int,res_inter);
        enable( );                                                         //开中断
}

void interrupt res_inter( )                                                //接收中断
{
        unsigned char temp;
//两个 SJA1000 共用一个中断,在程序中分别读取两个 SJA1000 的状态以确定是哪片接收到了数据
        disable( );                                                        //关中断

        /*can1 receive inter
        若接收到则将数据写入缓冲区 res1buf*/
        temp=sja_read(1,3);
        if(temp&0x08)                                                      //是否超载
        {
                sja_write(1,1,0x08);                                       //是，则清除超载状态
        }
```

```
	if(temp&0x01)                                   //若有接收中断
	{
		res1buf[1]=sja_read(1,22);          //则将数据写入缓冲区
		sja_write(1,1,0x0c);                //释放 TBF
		putchar(res1buf[1]);                //显示数据
		puts("=>Can1 Receive");
	}

	/*can2 receive inter
	若接收到则将数据写入缓冲区 res2buf*/
	temp=sja_read(2,3);
	if(temp&0x08)                                   //是否超载
	{
		sja_write(2,1,0x08);                //是，则清除超载状态
	}
	if(temp&0x01)                                   //若有接收中断
	{
		res2buf[1]=sja_read(2,22);          //则将数据写入缓冲区
		sja_write(2,1,0x0c);                //释放 TBF
		putchar(res2buf[1]);                //显示数据
		puts("=>Can2 Receive");
	}

	outportb(0x20,0x20);                            //发中断结束命令
	outportb(0xa0,0x20);
	enable( );                                      //开中断
}

void interrupt time_inter( )                            //定时检测 SJA1000 的状态(可省略)
{
	unsigned char temp;
	disable( );                                     //关中断

//定时检测 SJA1000 的状态，如：是否脱离总线等,以决定是否复位
	temp=sja_read(1,2);
	if((temp&0x80)= =0x80)
	{
		sja_write(1,0,0x01);                //若脱离，则将 SJA1000 复位
		sja_write(1,0,0x72);                //在进入正常工作方式
	}
	temp=sja_read(2,2);
	if((temp&0x80)= =0x80)
	{
		sja_write(2,0,0x01);                //若脱离，则将 SJA1000 复位
		sja_write(2,0,0x72);                //在进入正常工作方式
```

```
        }

enable( );                                              //开中断
}


void resume_inter( )                                    //恢复中断
{
        disable( );                                     //关中断
        setvect(use_int,old_res_inter);                 //恢复旧中断入口
        setvect(Int_time,old_time_inter);
        outportb(0x21,inter_ocw);                       //恢复旧中断屏蔽字
        outportb(0xa1,inter_ocw2);
        enable( );                                      //开中断
}

void reset_sja(int num)                                 //硬件复位 SJA1000
{
        if(num==1)                                      //复位 1 号 SJA1000
        {
                outport(sja_inf.cntr_add,0x4db2);
                delay(50);
                outport(sja_inf.cntr_add,0x4db6);
        }
        if(num==2)                                      //复位 2 号 SJA1000
        {
                outport(sja_inf.cntr_add,0x4d96);
                delay(50);
                outport(sja_inf.cntr_add,0x4db6);
        }
}

void enable_inter( )                                    //容许 SJA1000 中断
{
        outport(sja_inf.setinter,0x0049);
}
void disable_inter( )                                   //禁止 SJA1000 中断
{
        outport(sja_inf.setinter,0x0000);
}

//增加的函数
void help( )                                            //帮助信息
{
        printf("\n==================Test PCI_CAN==================");
```

```
        printf("\n'1' for can1 to can2, '2' for can2 to can1");
        printf("\n'3' for reset can1, '4' for reset can2");
        printf("\n'5' for initial can, '6' for enable interrupt");
        printf("\n'7' for disable interrupt , 'q' for exit.");
        printf("\n===============================================\n");
}

void send(int num,unsigned char aim_adress,unsigned char out_data)
{
        unsigned char temp;
        temp=sja_read(num,2);
        if((temp&0x04)==0x04)                      //TBF 是否可写
        {                                          //是，则写入报文
                sja_write(num,10,aim_adress);      //写目标地址
                sja_write(num,11,1);               //写发送字节数
                sja_write(num,12,out_data);        //写发送数据
                sja_write(num,1,0x01);             //启动发送
        }
}
```

6.4　PC 并行端口与 CAN 接口设计

6.4.1　PC 并行端口简介

每一台 PC 都至少配有一个与打印机相连的并行口，称之为打印机接口或 Centronics 接口，它是 PC 必备的标准接口之一。随着 PC 的发展和使用需求的不断拓展，并行口已由原来的单向传输变为双向传输，并且发展了与之相配合的传输协议和标准。IEEE 在这些标准的基础上作了进一步的改进和完善，并于 1994 年公布了 IEEE-1284 标准（IEEE-1284-994）。IEEE-1284 标准定义了 4 种工作模式（即 Centronics 兼容、半字节和字节、EPP、ECP）来支持目前所有的打印标准，并且增加了高速数据传输模式，制定了更好的物理连接规范和接口电气规范以及模式谈判协议。

通信卡在设计时采用了 EPP 模式，实现了数据的双向传输以及硬件握手协议。下面重点介绍并口（并行口）的 EPP 工作模式。

EPP（Enhanced Parallel Port）增强型并口除能实现双向数据传输之外，还扩展了以下几个方面的功能：

1）实现了硬件握手协议，即数据的输出/输入的握手工程由硬件自动完成，不再需要主机软件进行干预和控制。一方面提高了数据传输的可靠性，另一方面大大提高了并口的传输速度，使速度从标准并口的 200KB/s 提高到 2MB/s 左右。

2）在总线上可以传输 8bit 地址，允许总线连接多个不同设备或多功能设备，通过地址进行不同设备的选择。

3）EPP 扩充了标准并口的 I/O 口的地址，在原来 3 个 I/O 口地址的基础上新扩展了 5 个 I/O 口，一个用于地址传输，4 个用于数据传输并允许主机进行 32bit 的操作。

EPP 是一种定义比较完备的双向并口传输协议。

1．EPP 信号的定义

PC 机并行端口采用 DB25/Female（母）式插座，如图 6-23 所示。

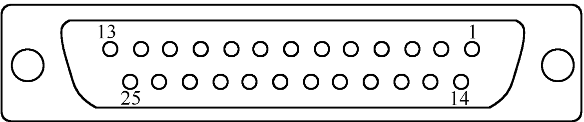


图 6-23　DB25 插座

各信号线的定义见表 6-2。

表 6-2　EPP 信号线的定义

引　脚	定　义	方　向	功 能 说 明
1	$\overline{\text{Write}}$	O	写：用于指示主机是向外设写（低电平）还是读（高电平）
11	Wait	I	等待：为低电平时表示外设准备好传输（读/写）数据，为高电平表示地址或数据的接收已经完成
10	Intr	I	中断请求：外设可通过此线的上升沿向主机申请中断，只要主机的中断是允许的（PCON 中的 D4 控制），将使主机产生中断 7 或 5
13	Xflag	I	有效表明外设处于联机状态
12	AckDataReq	I	响应数据请求：对主机发出的反向传输请求的响应，即外设接受反传请求
15	DataAvail	I	数据有效：向主机表明外设在总线上的数据有效
16	$\overline{\text{ReversRequest}}$	O	反传请求：主机控制此信号线表明总线上的传输方向，高电平表示正向输出，低电平表示反向输入
14	$\overline{\text{Dstrb}}$	O	数据选通：在写周期中，低有效表示主机向外设写数据且数据有效；在读周期中，此信号有效表示主机在等待读取外设的数据。此信号与 Wait 结合完成数据的双向传输。
2～9	PD［7：0］	I/O	数据：8bit 双向数据总线，可实现地址和数据的双向传输
17	$\overline{\text{Astrb}}$	O	地址选通：除了指示总线上的数据为地址外，其作用完全同数据选通。在写周期中，低有效表示主机向外设输出地址；在读周期中，此信号有效表示主机在等待外设输入地址。此信号与 Wait 结合完成地址的双向传输。
18～25	GND		信号地

2．EPP 寄存器

EPP 除了保留标准并口的 3 个寄存器 PDATA、PCON、PSTAT 之外，另外新增了 5 个寄存器：一个地址选通寄存器 ADDSTR 和 4 个数据选通寄存器 DATASTR，它需占用连续的 8 个端口地址。

（1）PDATA 数据寄存器

地址为基址 378H 或 278H。当 PCON 中的方向位 DIR＝0 时为输出，写入此寄存器的数据直接输出到 PD[7：0]数据总线上；当 DIR＝1 时为输入，主机读 PDATA 将 PD[7：0]总线上的数据读入。

（2）PCON 控制寄存器

地址为基址+02H。PCON 中的各位含义如图 6-24 所示。

（3）PSTAT 状态寄存器

地址为基址+01H。PSTAT 中的各位含义如图 6-25 所示。

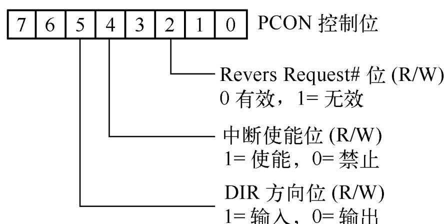


图 6-24　PCON 各位含义

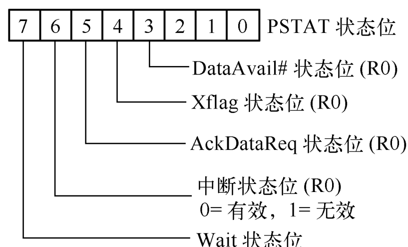


图 6-25　PSTAT 各位含义

（4）ADDSTR（Auto Address Strobe Register）地址选通寄存器

地址为基址+03H。主机往此口写入数据将在总线上激发一个自动地址传输周期（$\overline{\text{Astrb}}$有效），此口的地址通过 PD[7：0]输出到外设。当主机读此口时将产生一个自动地址读周期，外设向 PD[7：0]上输出的地址值从 ADDSTR 中读入主机。

（5）DATASTR（Auto Data Strobe Register）数据选通寄存器

共有 4 个，地址为基址+04H、05H、06H、07H。对 DATASTR 的读写操作过程完全同对 PDATA 的操作，用于实现主机对外设的数据输出/输入，并自动产生$\overline{\text{Dstrb}}$信号。这 4 个寄存器可以单独使用，每次自动传输一个字节；也可以合成一次 32bit 的读/写，并自动产生 4 个读/写数据传输周期，主要是为配合 32bit 数据总线的传输而设置的。

3．EPP 读写时序

EPP 的写时序如图 6-26 所示。

EPP 的工作过程如下：当主机向外设输出地址或数据时，首先使 PCON 中的方向位为 0（输出），然后向 PDATA（或 DATASTR）或 ADDSTR 寄存器写入数据即可，EPP 将自动产生$\overline{\text{Dstrb}}$或$\overline{\text{Astrb}}$选通信号，同时产生总线方向传输信号$\overline{\text{Write}}$，写周期何时完成取决于外设的状态，如果外设正忙（Wait 高电平）则主机一直处于等待状态，并一直等到 Wait 为低电平（即，外设接收数据）为止。外设将数据读入后再次使 Wait 为高，表明数据已接收到，此时 EPP 才结束本次数据的输出过程。

EPP 的读时序如图 6-27 所示。EPP 的工作过程如下：当 EPP 进行数据/地址输入时，首先使 PCON 中的方向位为 1（输入），然后读数据寄存器或地址寄存器即可，EPP 将自动产生$\overline{\text{Dstrb}}$或$\overline{\text{Astrb}}$并且通过使$\overline{\text{Write}}$无效（高电平）告之外设为读数据/地址操作，如果外设正忙，即 Wait 为高电平时为等待状态，直至 Wait 取消（为低）表明外设正在准备数据，数据准备好且已稳定有效时使 Wait 为高，告诉主机输入数据已准备好，然后将数据读入并结束本次输入过程。

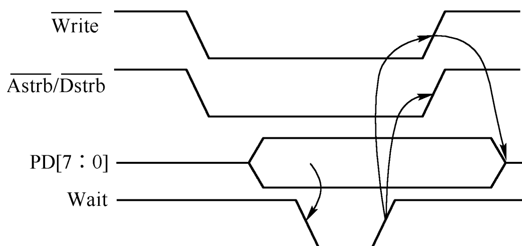


图 6-26　EPP 写时序图

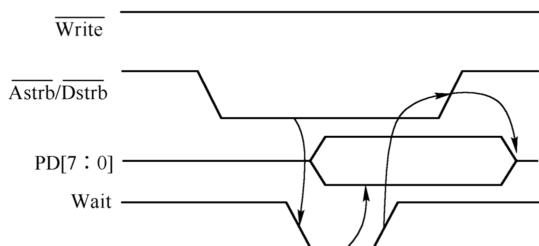


图 6-27　EPP 读时序图

可见，在整个传输过程中，EPP 不需要软件实现握手过程。

6.4.2 基于 EPP 模式的接口电路设计

CAN 总线控制器选用了 SJA1000，收发器选用了 82C250。为了保证使用中 PC 的安全，加入了 DC/DC 电源隔离器和光耦隔离，使 CAN 总线在电气上完全与 PC 隔离。所以接口电路的设计主要包括 SJA1000 与并行端口的接口以及 SJA1000 与 82C250 的接口两个部分。

并口与 SJA1000 的接口电路如图 6-28 所示。

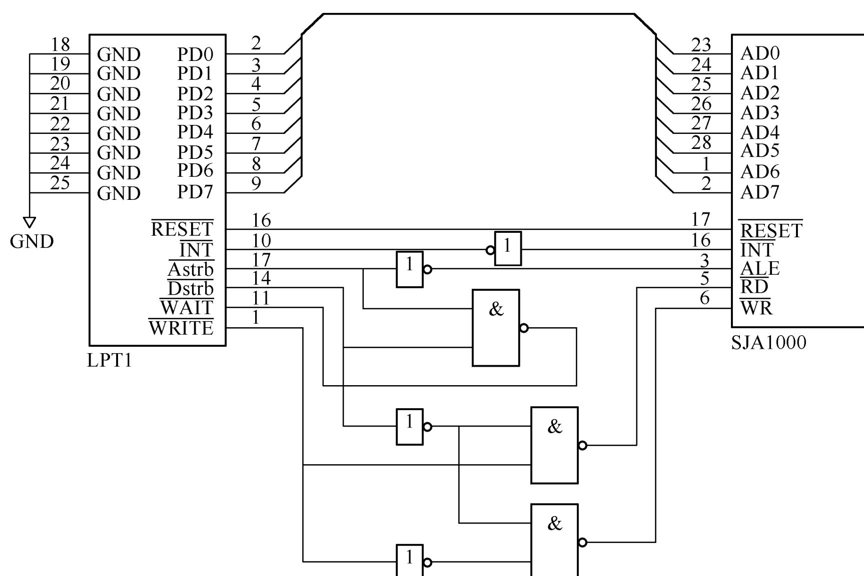


图 6-28 并口与 SJA1000 接口电路

SJA1000 的 AD0～AD7 分别与并口的 PD0～PD7 相连，构成双向地址/数据线。因为 SJA1000 的访问时间在 40ns 以内，所以 PC 在访问 SJA1000 时不需要插入等待周期。Wait 信号由 $\overline{\text{Astrb}}$ 和 $\overline{\text{Dstrb}}$ 或通过一个与非门生成。使用并口的 $\overline{\text{ReversRequest}}$ 控制 SJA1000 的复位，SJA1000 的中断信号通过非门将 SJA1000 的低电平中断信号转化为上升沿向主机申请中断。SJA1000 的地址锁存信号 ALE 由 $\overline{\text{Astrb}}$ 经过非门产生。$\overline{\text{RD}}$ 和 $\overline{\text{WR}}$ 是通过 $\overline{\text{Dstrb}}$ 和 $\overline{\text{Write}}$ 共同合成的。SJA1000 的片选信号 $\overline{\text{CS}}$ 始终接地，使其一直处于选通状态。

对 SJA1000 的一次读/写操作是通过并口的一次地址写操作和一次数据读/写操作合成的。首先进行一次并口的写地址操作，向 SJA1000 写入片内寄存器的地址。然后再进行并口的数据读/写，完成并口与 SJA1000 的数据通信。

CAN 网络输出接口的硬件设计与 6.1.2 节基本相同，采取了光耦隔离的保护措施。并行端口 CAN 接口卡总电路原理图，如图 6-29 所示。

6.4.3 并口接口卡软件设计

软件是使用 C 语言编写的，主要完成两个方面的工作。首先是编写并口与 SJA1000 的通信程序，即通过并口完成对 SJA1000 的读写以及对中断的响应，其次是 CAN 总线通信程序，包括 SJA1000 的初始化、发送报文、接收报文以及对一般错误的处理。

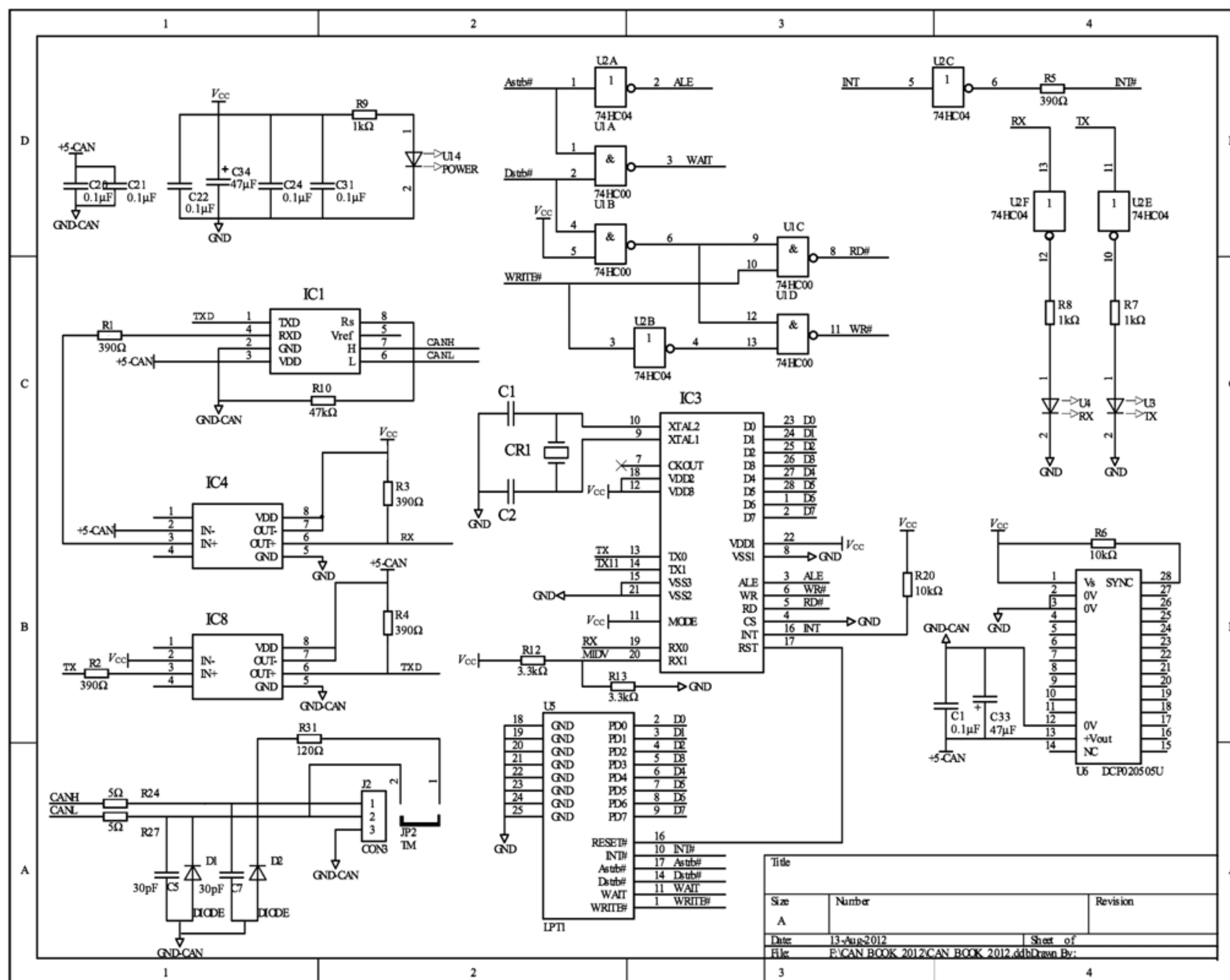


图 6-29　并行端口接口卡电路原理图

1．并口与 SJA1000 的通信

（1）通过并口读 SJA1000

```
#define Basic_Adress 0x378                              //定义基地址
/*参数：in_adress－SJA1000 内部寄存器地址
            返回值－读出的数据*/
unsigned char SJA_Read(unsigned char in_adress)         //读 SJA1000
{
        unsigned char in_data,temp;
        outportb(Basic_Adress+3,in_adress);             //写入片内寄存器地址
        in_data=inportb(Basic_Adress+4);                //从 SJA1000 读出数据
        return(in_data);                                //返回数据
}
```

（2）通过并口写 SJA1000

```
/*参数：in_adress－SJA1000 内部寄存器地址
                              out_data－要写入的数据*/
void SJA_Write(unsigned char in_adress,unsigned char out_data)
{
        unsigned char temp;
        outportb(Basic_Adress+3,in_adress);             //写入片内寄存器地址
        outportb(Basic_Adress+4,out_data);              //向 SJA1000 写入数据
}
```

（3）复位 SJA1000

```
void reset_can( )                                       //复位 SJA1000
{
        unsigned char temp;
        temp=inportb(Basic_Adress+2);
        temp=temp&0xfb;
        outportb(Basic_Adress+2,temp);
        delay(500);
        temp=temp|0x04;
        outportb(Basic_Adress+2,temp);
}
```

（4）并口中断的使能与禁止

```
void en_inter( )                                        //中断使能
{
        unsigned char temp；

        temp=inportb(Basic_Adress+2)；
        temp=temp|0x10；
        outportb(Basic_Adress+2，temp)；
}
```

```
void dis_inter( )                                   //中断禁止
{
        unsigned char temp；

        temp=inportb(Basic_Adress+2);
        temp=temp&0xef;
        outportb(Basic_Adress+2，temp);
}
```

（5）装载中断程序

有两个中断需要处理，一个是 SJA1000 接收中断；另一个是为了保证系统的可靠性需要定时检测 SJA1000 的状态，并做出相应的处理。所以中断程序包括对外中断 IRQ7 的处理和定时产生一个软件中断。中断程序安装时需要设置新的中断入口，退出时必须恢复原中断入口。

```
#define Int_num 0x0f                          //中断向量号 IRQ7 0x0f
#define Int_time 0x1c                         //定时中断向量号
void interrupt (*old_can_inter)( );           //保存旧中断入口
void interrupt (*old_time_inter)( );

void install_inter( )                         //装载中断
{
        disable( );                           //关中断
        old_can_inter=getvect(Int_num);       //保存旧中断入口
        old_time_inter=getvect(Int_time);
        setvect(Int_num,can_inter);           //设置新中断入口
        setvect(Int_time,time_inter);
        enable( );                            //开中断
}

void resume_inter( )                          //恢复中断
{
        disable( );                           //关中断
        setvect(Int_num,old_can_inter);       //恢复旧中断入口
        setvect(Int_time,old_time_inter);
        outportb(0x21,inter_ocw);             //恢复旧中断屏蔽字
        outportb(Basic_Adress+2,pl_con);
        enable( );                            //开中断
}
inter_ocw=inportb(0x21);                      //保存旧中断屏蔽字
outportb(0x21,inter_ocw&0x00);                //写中断屏蔽字，打开 IRQ5、IRQ7
```

2．CAN 总线通信

（1）SJA1000 的初始化

```
#define    Local_1     1                      //验收报文 ID
void initial_can( )                           //初始化 SJA1000
```

```
{
        disable( );
        //initial can
        SJA_Write(0,0x01);
        SJA_Write(4,Local_1);                    //验收报文 ID
        SJA_Write(5,0x03);
        SJA_Write(6,0x47);
        SJA_Write(7,0x2f);
        SJA_Write(8,0xda);
        SJA_Write(0,0x72);
        enable( );
}
```

（2）发送报文

```
//发送数据，参数：aim_adress－目标地址；out_data－要发送的数据
void send(unsigned char aim_adress,unsigned char out_data)
{
        unsigned char temp;

        temp=SJA_Read(2);
        if((temp&0x04)==0x04)                    //TBF 是否可写
        {                                        //是，则写入报文
        SJA_Write(10,aim_adress);                //写报文 ID
        SJA_Write(11,1);                         //写发送字节数
        SJA_Write(12,out_data);                  //写发送数据
        SJA_Write(1,0x01);                       //启动发送
        }
}
```

（3）中断接收报文

```
void interrupt can_inter( )                      //CAN 接收中断
{
        unsigned char temp;
        disable( );                              //关中断

        temp=SJA_Read(3);
        if(temp&0x08)                            //是否超载
        {
                SJA_Write(1,0x08);               //是，则清除超载状态
        }
        if(temp&0x01)                            //若有接收中断
        {
                canbuf[1]=SJA_Read(22);          //则将数据写入缓冲区
                SJA_Write(1,0x0c);               //释放 TBF
                re_key=1;
        }
        outportb(0x20,0x20);                     //发中断结束命令
```

```
        enable( );                                          //开中断
    }
```

（4）一般错误处理

```
    void interrupt time_inter( )                            //定时检测 SJA1000 是否总线脱离
    {
        unsigned char temp;
        disable( );                                         //关中断

        temp=SJA_Read(2);
        if((temp&0x80)==0x80)
        {
            SJA_Write(0,0x01);                              //若脱离，则将 SJA1000 复位
            SJA_Write(0,0x72);                              //在进入正常工作方式
        }
        enable( );                                          //开中断
    }
```

注意：运行程序前请进入电脑的 BIOS 设置，将并口的工作方式设置为 EPP 方式，地址设置为 0x378，中断号设置为 IRQ7。通信测试的完整源程序如下所示。

```
/*******************************************************************************
    基于并口的 CAN 网通信卡的配套测试程序
    编程语言：C 语言
    运行环境：DOS
    接收方式：中断式接收
    注意：运行程序前请进入电脑的 BIOS 设置，将并口的工作方式设置为 EPP 方式，
          地址设置为 0x378，中断号设置为 IRQ7。
*******************************************************************************/

#include "stdio.h"
#include "dos.h"
#include "bios.h"

#define Basic_Adress 0x378                                  //基地址
#define Int_num 0x0f                                        //中断向量号 IRQ7 0x0f
#define Int_time 0x1c                                       //定时中断向量号
#define   Local_1    1                                      //CAN 的本站地址
#define   Local_2    2

void interrupt can_inter( );                                //CAN 接收中断
void interrupt time_inter( );                               //定时中断
void interrupt (*old_can_inter)( );                         //保存旧中断入口
void interrupt (*old_time_inter)( );
void install_inter( );                                      //装载中断
void resume_inter( );                                       //恢复中断
```

```
void initial_can( );                                    //初始化 SJA1000
unsigned char pl_con,re_key;

/*写 SJA1000，参数：      in_adress－SJA1000 内部寄存器地址
                         out_data－要写入的数据*/
void SJA_Write(unsigned char in_adress,unsigned char out_data);

/*读 SJA1000，参数：      in_adress－SJA1000 内部寄存器地址
                         返回值－读出的数据*/
unsigned char SJA_Read(unsigned char in_adress);

/*发送数据，参数：        aim_adress－目标地址
                         out_data－要发送的数据*/
void send(unsigned char aim_adress,unsigned char out_data);

//复位 SJA1000
void reset_can( );

//中断使能
void en_inter( );

//中断禁止
void dis_inter( );

void help( );      //帮助信息

unsigned char canbuf[10],inter_ocw;     //全局变量：canbuf-输入缓存；inter_ocw－中断屏蔽字

main( )
{
        unsigned char u_data;
        char in_key;
        unsigned long i;

        re_key=0;
        pl_con=inportb(Basic_Adress+2);
        outportb(Basic_Adress+2,0x00);          //屏蔽中断，复位 SJA1000
        delay(500);
        outportb(Basic_Adress+2,0x14);          //屏蔽中断，恢复 SJA1000

        initial_can( );                         //初始化 SJA1000
        install_inter( );                       //装载中断
        inter_ocw=inportb(0x21);                //保存旧中断屏蔽字
        outportb(0x21,inter_ocw&0x00);          //写中断屏蔽字，打开 IRQ7
        enable( );
        help( );                                //输出帮助信息
```

```
        do{
            while(!bioskey(1))
            {
                if(re_key==1)
                {
                    re_key=0;
                    putchar(canbuf[1]);     //显示数据
                    puts("=>Can Receive");
                }
            }
            in_key=getch( );                //输入选择
            switch(in_key)
            {
            case '1':                       //输入'1'－CAN 向外发送数据
                printf("Can Send=");
                u_data=getch( );            //输入要发送的数据
                printf("%c\n",u_data);
                send(1,u_data);
                break;
            case '2':                       //输入'2'－复位 SJA1000
                printf("Reset Can.\n");
                reset_can( );
                break;
            case '3':                       //输入'3'－初始化 SJA1000
                printf("Initial Can.\n");
                initial_can( );
                break;
            case '4':                       //输入'4'－中断使能
                printf("Enable interrupt.\n");
                en_inter( );
                break;
            case '5':                       //输入'5'－中断禁止
                printf("Disenable interrupt.\n");
                dis_inter( );
                break;
            case 'q':                       //输入'q'－结束程序
                resume_inter( );
                printf("Test over.\n");
                break;
            default:                        //输入其他值显示帮助信息
                help( );
        }
    }while(in_key!='q');
}
```

```
void help( )                                              //帮助信息
{
        printf("\n===============Test PL_CAN================\n");
        printf("'1' for can send data , '2' for reset can\n");
        printf("'3' for initial can, '4' for Enable interrupt\n");
        printf("'5' for Disenable interrupt.\n");
        printf("'q' for exit.\n");
        printf("===========================================\n");
}

void initial_can( )                                       //初始化 SJA1000
{
        disable( );
        //initial can
        SJA_Write(0,0x01);
        SJA_Write(4,Local_1);                             //站地址
        SJA_Write(5,0x03);
        SJA_Write(6,0x47);
        SJA_Write(7,0x2f);                                //设置波特率
        SJA_Write(8,0xda);                                //推挽输出
        SJA_Write(0,0x72);
        enable( );
}

void send(unsigned char aim_adress,unsigned char out_data)
{
        unsigned char temp;

        temp=SJA_Read(2);
        if((temp&0x04)==0x04)                             //TBF 是否可写
        {                                                 //是，则写入报文
            SJA_Write(10,aim_adress);                     //写目标地址
            SJA_Write(11,1);                              //写发送字节数
            SJA_Write(12,out_data);                       //写发送数据
            SJA_Write(1,0x01);                            //启动发送
        }
}

void SJA_Write(unsigned char in_adress,unsigned char out_data)//写 SJA1000
{
        unsigned char temp;

        outportb(Basic_Adress+3,in_adress);               //写入片内寄存器地址
        outportb(Basic_Adress+4,out_data);                //向 SJA1000 写入数据
}
```

```
unsigned char SJA_Read(unsigned char in_adress)    //读 SJA1000
{
        unsigned char in_data,temp;

        outportb(Basic_Adress+3,in_adress);     //写入片内寄存器地址
        in_data=inportb(Basic_Adress+4);        //从 SJA1000 读出数据
        return(in_data);                        //返回数据
}

void reset_can( )                               //复位 SJA1000
{
        unsigned char temp;

        temp=inportb(Basic_Adress+2);
        temp=temp&0xfb;
        outportb(Basic_Adress+2,temp);
        delay(500);
        temp=temp|0x04;
        outportb(Basic_Adress+2,temp);
}

void en_inter( )        //中断使能
{
        unsigned char temp;

        temp=inportb(Basic_Adress+2);
        temp=temp|0x10;
        outportb(Basic_Adress+2,temp);
}

void dis_inter( )  //中断禁止
{
        unsigned char temp;

        temp=inportb(Basic_Adress+2);
        temp=temp&0xef;
        outportb(Basic_Adress+2,temp);
}

void install_inter( )      //装载中断
{
        disable( );  //关中断
        old_can_inter=getvect(Int_num);         //保存旧中断入口
        old_time_inter=getvect(Int_time);
        setvect(Int_num,can_inter);             //设置新中断入口
        setvect(Int_time,time_inter);
```

```
        enable( );                          //开中断
}

void interrupt can_inter( )                 //CAN 接收中断
{
        unsigned char temp;
        disable( );                         //关中断

        temp=SJA_Read(3);
        if(temp&0x08)                       //是否超载
        {
            SJA_Write(1,0x08);              //是，则清除超载状态
        }
        if(temp&0x01)                       //若有接收中断
        {
            canbuf[1]=SJA_Read(22);         //则将数据写入缓冲区
            SJA_Write(1,0x0c);              //释放 TBF
            re_key=1;
        }
        outportb(0x20,0x20);                //发中断结束命令
        enable( );                          //开中断
}

void interrupt time_inter( )                //定时检测 SJA1000 是否总线脱离
{
        unsigned char temp;
        disable( );                         //关中断

        temp=SJA_Read(2);
        if((temp&0x80)= =0x80)
        {
            SJA_Write(0,0x01);              //若脱离，则将 SJA1000 复
            SJA_Write(0,0x72);              //在进入正常工作方式
        }
        enable( );                          //开中断
}

void resume_inter( )                        //恢复中断
{
        disable( );                         //关中断
        setvect(Int_num,old_can_inter);     //恢复旧中断入口
        setvect(Int_time,old_time_inter);
        outportb(0x21,inter_ocw);           //恢复旧中断屏蔽字
        outportb(Basic_Adress+2,pl_con);
        enable( );                          //开中断
}
```

6.5　USB 总线与 CAN 接口设计

6.5.1　USB 总线简介

USB 是英文 Universal Serial BUS（通用串行总线）的缩写，应用于 PC 与外部设备的连接和通信。USB 是在 1994 年底由英特尔、康柏、IBM、Microsoft 等多家公司联合提出的。USB 接口成为目前电脑中的标准扩展接口，逐渐淘汰了 RS-232 串口和并行接口。

USB 用一个 4 针插头作为标准插头，采用菊花链形式可以把所有的外设连接起来，最多可以连接 127 个外部设备，并且不会损失带宽。其中两个插头由 PC 提供给外设 5V 电源，另两个插头为差分信号线。USB 包含了控制传输、批量传输、中断传输、同步传输 4 种基本的数据传输类型。

PC 控制端口初始化所有的数据传输。每一总线动作最多传送 3 个数据包，包括令牌(Token)、数据(Data)、联络(HandShake)。按照传输前制定好的原则，在每次传送开始时，PC 传送一个描述传输动作的种类、方向、USB 设备地址和终端号的 USB 数据包，这个数据包通常被称为令牌包(TokenPacket)。USB 设备从解码后的数据包的适当位置取出属于自己的数据。数据传输方向不是从主机到设备就是从设备到主机。在传输开始时，由标志包来标记数据的传输方向，然后发送端开始发送包含信息的数据包或表明没有数据传送。接收端也要相应发送一个握手的数据包表明是否传送成功。

与 PCI 总线类似，USB 总线给用户的使用带来了极大的方便，逐渐成为 PC 最主要的外设接口之一。但这同时也增加了开发者的难度。USB 通信协议复杂，为了实现即插即用等功能，在设备接入后有一个复杂的配置和通道建立的过程，无法使用简单的逻辑电路实现通信。在此选用“USB-串口”转换芯片 FT232BM，实现与 USB 总线的接口。FT232BM 实现了 USB 总线的枚举、资源分配等功能，配置完成后在 PC 上建立了一个虚拟的串口设备，同时为本地提供一个异步串行总线，这样就避免了复杂的 USB 通信协议。

6.5.2　硬件电路设计说明

在使用了 FT232BM 接口芯片之后，对 PC 而言，适配卡就是一个标准的串口设备。在软件编程上完全等同于对标准串口的操作。而对本地设备而言，FT232BM 提供了一个标准的异步串行总线。这样就建立了一个 PC 和本地设备之间的异步串行通信通道，可以认为中间转换过程是透明的，降低了设计难度。重点解决本地串口与 CAN 总线之间的转换关系即可。在此使用单片机完成本地串行总线的通信以及对 CAN 总线接口芯片 SJA1000 的控制。硬件结构如图 6-30 所示。

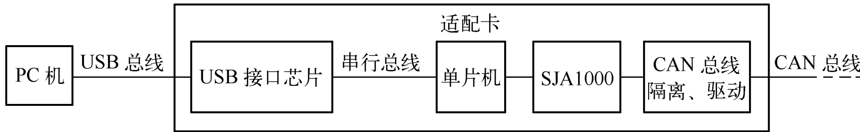


图 6-30　USB 总线适配卡硬件结构图

USB 接口芯片的硬件设计如图 6-31 所示。

从图 6-31 可以看出，适配卡从“USB 总线”获取 5V 电源。为了保护 PC 的安全，在电

源输入端串联了一个 0.4A 的熔断器。FT232BM 提供了一个本地串口（TXD/RXD），以及一对收发指示灯（TXLED/RXLED）。

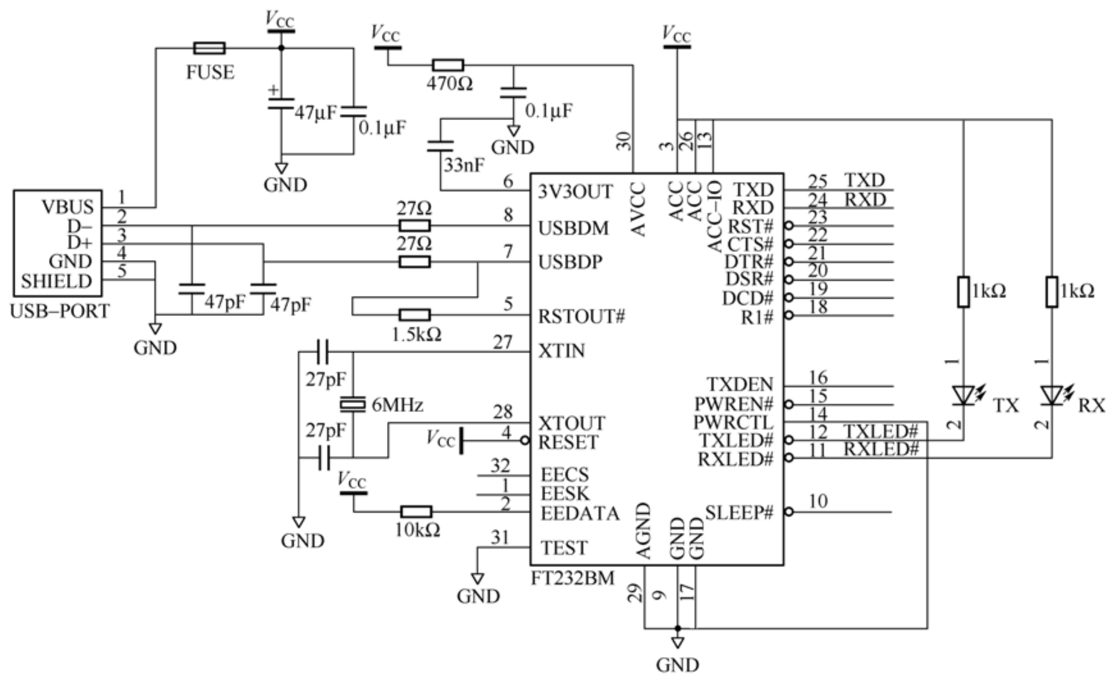


图 6-31　FT232BM 硬件设计图

单片机与 SJA1000 之间的硬件连接以及 CAN 总线接口隔离等设计与 6.1.2 节基本相同，只是在单片机上增加了一个与 FT232BM 之间的串行接口。基于 USB 总线的 CAN 接口卡电路原理图如图 6-32 所示。

6.5.3　USB 接口卡软件设计

适配卡的编程分为两个部分，一是适配卡上单片机的程序编写，实现串口数据收发、打包、错误识别以及对 SJA1000 的操作等；二是 PC 上位程序的编写，实现虚拟串口数据的收发、打包、错误识别以及显示等功能。

由于 CAN 总线的数据是以数据帧为单位进行传输的，而串行总线是以字节为传输单位的。所以需要将串行总线的数据进行打包处理，以 10B（8bit）为一帧，第一个字节为帧头，末一个字节为帧尾，中间 8B 为要传输的数据。帧头表示帧类型，帧尾为前 9 个数据的和（超出 8bit 的舍弃），实现错误判别。

打包的帧类型主要有控制帧、数据帧、特殊帧等。控制帧实现对适配卡的设置，如复位、CAN 总线波特率设定、报文 ID 设定等，主要由 PC 提供完成 SJA1000 初始化的一些参数；数据帧为要传输的有效数据；特殊帧是表示 CAN 总线传输中一些特殊的数据帧，如远程帧、0B 长度的数据帧等。

单片机的编程首先实现数据的分类、打包、串行通信、错误处理等功能，实现和 FT232BM 之间的接口协议。其次完成 SJA1000 的初始化、CAN 总线数据收发、错误处理等功能。单片机的源程序如下。

图 6-32　USB 总线接口卡电路原理图

```
#include <reg51.h>
//定义 IO 端口
sbit SJA_ALE    = P3^3;
sbit SJA_RD     = P3^4;
sbit SJA_WR     = P3^5;
sbit SJA_INT    = P3^2;
sbit SJA_RST    = P3^7;

unsigned char Command,Verify,Read_Address,Write_Address,Write_Data;//Intter_Control;
unsigned char CRead_Address,CRead_Num,CWrite_Address,CWrite_Num,DBUF[25],FLAG;

//外中断
void INT_0(void) interrupt 0
{
        EA=0;
        SBUF=0x55;while(!TI);TI=0;
        EA=1;
}
//初始化 IO 端口
void Init_Port( )
{
        SJA_ALE = 0;
        SJA_RD = 1;
        SJA_WR = 1;
        SJA_INT = 1;
        SJA_RST = 1;
}
//写 SJA1000
void SJA_Write(unsigned char address,unsigned char wr_data)
{
        P1= address;
        SJA_ALE = 1;
        SJA_ALE = 0;

        P1= wr_data;
        SJA_WR= 0;
        SJA_WR= 1;
}
//读 SJA1000
unsigned char SJA_Read(unsigned char address)
{
        unsigned char temp;
        P1= address;
        SJA_ALE = 1;
        SJA_ALE = 0;
```

```
        P1= 0xFF;
        SJA_RD = 0;
        temp= P1;
        SJA_RD = 1;

        return (temp);
}
//复位 SJA1000
void SJA_Reset(void)
{
        SJA_RST = 0;
        //DelayMs(1);
        SJA_RST = 1;
}
//串口接收中断
void UART(void) interrupt 4
{
        unsigned long cnt;
        unsigned char add,i,effect;

        EA=0;
        effect=0x00;
        if(RI==0) {EA=1;return;}

        Command=SBUF;
        RI=0;

        //选择进行操作 Read INT/control interrupt/Reset SJA
        if((Command == 0x01) || (Command == 0x02) || (Command == 0x04) ||   (Command == 0x08))
        {
            add = Command;
            effect=0xFF;
      }
        else if(Command == 0x10)                    //读数据
      {
            cnt=0;
            while((!RI)&&(cnt < 0xFFF)) ++cnt;
            if(cnt>=0xFFF) {EA=1;return;}
            Read_Address = SBUF;
            RI=0;
            add = Command+Read_Address;
            effect=0xFF;
      }
        else if(Command == 0x20)                    //写数据
        {
            cnt=0;
```

```
    while((!RI)&&(cnt < 0xFFF)) ++cnt;
    if(cnt>=0xFFF) {EA=1;return;}
    Write_Address = SBUF;
    RI=0;

    add = Command+Write_Address;

    cnt=0;
    while((!RI)&&(cnt < 0xFFF)) ++cnt;
    if(cnt>=0xFFF) {EA=1;return;}
    Write_Data = SBUF;
    RI=0;

    add = add+Write_Data;

    effect=0xFF;
}
else if(Command = = 0x40)                    //连续读
{
    cnt=0;
    while((!RI)&&(cnt < 0xFFF)) ++cnt;
    if(cnt>=0xFFF) {EA=1;return;}
    CRead_Address = SBUF;
    RI=0;

    add = Command+CRead_Address;

    cnt=0;
    while((!RI)&&(cnt < 0xFFF)) ++cnt;
    if(cnt>=0xFFF) {EA=1;return;}
    CRead_Num = SBUF;
    RI=0;

    add = add+CRead_Num;

    effect=0xFF;
}
else if(Command = = 0x80)                    //连续写
{
    cnt=0;
    while((!RI)&&(cnt < 0xFFF)) ++cnt;
    if(cnt>=0xFFF) {EA=1;return;}
    CWrite_Address = SBUF;
    RI=0;

    add = Command+CWrite_Address;
```

```
            cnt=0;
            while((!RI)&&(cnt < 0xFFF)) ++cnt;
            if(cnt>=0xFFF) {EA=1;return;}
            CWrite_Num = SBUF;
            RI=0;

            add = add+CWrite_Num;

            if(CWrite_Num < 21)
            {
                for(i=0;i<CWrite_Num;i++)
                {
                    cnt=0;
                    while((!RI)&&(cnt < 0xFFF)) ++cnt;
                    if(cnt>=0xFFF) {EA=1;return;}
                    DBUF[i]=SBUF;
                    RI=0;
                    add=add+DBUF[i];
                }
                effect=0xFF;
            }
        }

        if(effect)
        {
            cnt=0;
            while((!RI)&&(cnt < 0xFFF)) ++cnt;
            if(cnt>=0xFFF) {EA=1;return;}
            Verify = SBUF;
            RI=0;
            if(add == Verify) FLAG = FLAG | Command;
        }
        RI=0;EA=1;return;
}
//初始化串口
void Init_COMM(void)
{
        SCON = 0x50;
        PCON = 0x80 | PCON;
        TMOD=0x20;
        TL1=0xFF;
        TH1=0xFF;//115200
        TI=0;
        RI=0;
        TR1=1;
```

```
        ES = 1;
}
void main(void)
{
        unsigned char temp,i,add;

        IE=0x00;
        Init_Port( );
        SJA_Reset( );
        Init_COMM( );
        FLAG=0x00;
        IT0 = 1;
        EA=1;

        while(1)
        {
            if(FLAG)
            {
                if(FLAG & 0x01)        //读 INT0
                {
                    EA = 0; REN = 0;
                    temp = 0x00;temp = temp | SJA_INT;
                    SBUF = temp;while(!TI);TI = 0;
                    SBUF = temp;while(!TI);TI = 0;
                    EA = 1;REN = 1;
                    FLAG = FLAG & 0xFE;
                }
                if(FLAG & 0x02)        //使能 INT0
                {
                    EA = 0; REN = 0;
                    EX0 = 1;
                    SBUF = 0x02;while(!TI);TI = 0;
                    EA = 1; REN = 1;
                    FLAG = FLAG & 0xFD;
                }
                if(FLAG & 0x04)        //禁止 INT0
                {
                    EA = 0; REN = 0;
                    EX0 = 0;
                    SBUF = 0x04;while(!TI);TI = 0;
                    EA = 1; REN = 1;
                    FLAG = FLAG & 0xFB;
                }
                if(FLAG & 0x08)        //复位 SJA1000
                {
                    SJA_Reset( );
```

```
        EA = 0; REN = 0;
        SBUF = 0x08;while(!TI);TI = 0;
        EA = 1; REN = 1;
        FLAG = FLAG & 0xF7;
    }
    if(FLAG & 0x10)        //读数据
    {
        EA = 0; REN = 0;
        temp=SJA_Read(Read_Address);
        SBUF=temp;while(!TI);TI=0;
        SBUF=temp;while(!TI);TI=0;
        EA = 1; REN = 1;
        FLAG = FLAG & 0xEF;
    }
    if(FLAG & 0x20)        //写数据
    {
        SJA_Write(Write_Address,Write_Data);
        EA = 0; REN = 0;
        SBUF = 0x20;while(!TI);TI = 0;
        EA = 1; REN = 1;
        FLAG = FLAG & 0xDF;
    }
    if(FLAG & 0x40)        //连续读
    {
        EA = 0; REN = 0;
        add=0x00;
        for(i=0;i<CRead_Num;i++)
        {
            temp=SJA_Read(CRead_Address+i);
            SBUF=temp;
            add=add+temp;
            while(!TI);TI=0;
        }
        SBUF=add;while(!TI);TI=0;
        EA = 1; REN = 1;
        FLAG = FLAG & 0xBF;
    }
    if(FLAG & 0x80)        //连续写
    {
        for(i=0;i<CWrite_Num;i++)
        {
            SJA_Write(CWrite_Address+i,DBUF[i]);
        }

        EA = 0; REN = 0;
        SBUF = 0x80;while(!TI);TI = 0;
```

```
                    EA = 1; REN = 1;
                    FLAG = FLAG & 0x7F;
                }
            }
        }

    }
```

PC 的编程采用 C++语言，使用 VC 编译工具，主要实现数据分类、打包、串行通信、错误处理，以及输入输出等人机交互功能。

当适配卡接入 PC 后，PC 会检测到该设备，并要求安装驱动程序。在此使用 FT232BM 厂商提供的驱动程序完成设备安装，安装完成后 PC 设备管理中会显示一个虚拟的 USB-串行端口，如图 6-33 所示。



图 6-33　PC 设备管理器

如上图所示，串行端口的标号为 COM4，编程中只需对 COM4 端口进行操作即可。在此简单介绍一下实现串口初始化、数据发送、数据接收等功能的程序，对于显示等人机交互的程序可参考 C++编程的相关书籍。

端口 COM4 的建立和初始化：

```
HANDLE      hPort;                          //定义句柄
DCB         PortDCB;                        //端口设置结构体
//打开 COM4 端口并获得一个文件句柄，对 COM4 的操作等同于对一个文件的操作
hPort = CreateFile("COM4",GENERIC_READ|GENERIC_WRITE,0, NULL,OPEN_EXISTING,
```

```
FILE_ATTRIBUTE_NORMAL,NULL);
if(INVALID_HANDLE_VALUE = = hPort)          //检查是否成功打开端口
{
        MessageBox("系统故障，无法打开设备!请先安装驱动程序并正确设置串口号。");
        return;
}
PortDCB.DCBlength = sizeof(DCB);
if(!GetCommState(hPort,&PortDCB))           //获取端口状态
{
        CloseHandle(hPort);
        MessageBox("系统故障，无法打开设备!请先安装驱动程序并正确设置串口号。");
        return;
}
PortDCB.fBinary = TRUE;
PortDCB.fOutxCtsFlow = FALSE;
PortDCB.fOutxDsrFlow = FALSE;
PortDCB.fDtrControl = DTR_CONTROL_ENABLE;
PortDCB.fRtsControl = RTS_CONTROL_ENABLE;
PortDCB.fOutX = FALSE;
PortDCB.fInX =FALSE;
//以下参数需要与单片机相关设置相一致
PortDCB.BaudRate = 115200;                 //波特率
PortDCB.ByteSize = 8;                      //8bit 数据位
PortDCB.Parity = NOPARITY;                 //无奇偶校验位
PortDCB.StopBits = ONESTOPBIT;             //1 个停止位

if(!SetCommState(hPort,&PortDCB))           //设置端口参数
{
        CloseHandle(hPort);
        MessageBox("系统故障，无法打开设备!请先安装驱动程序并正确设置串口号。");
        return;
}
COMMTIMEOUTS CommTimeouts;
GetCommTimeouts(hPort,&CommTimeouts);
CommTimeouts.ReadIntervalTimeout = 10;
CommTimeouts.ReadTotalTimeoutMultiplier = 10;
CommTimeouts.ReadTotalTimeoutConstant = 100;
CommTimeouts.WriteTotalTimeoutMultiplier = 10;
CommTimeouts.WriteTotalTimeoutConstant = 100;
//设定发送和接收缓冲区
if((!SetCommTimeouts(hPort,&CommTimeouts))&&(!SetupComm(hPort,256,256)))
      {
        CloseHandle(hPort);
        MessageBox("系统故障，无法打开设备!请先安装驱动程序并正确设置串口号。");
        return;
}
```

```
PurgeComm(hPort,PURGE_TXCLEAR);                    //清空发送缓冲区
PurgeComm(hPort,PURGE_RXCLEAR);                    //清空接收缓冲区
```

数据发送，以发送 10B 的数据为例：

```
unsigned char TxData[10];
    DWORD TxNum;
    TxData[0]=0x00;
…
    TxData[9]=0x09;
    WriteFile(hPort,TxData,10,&TxNum,NULL);
    if(TxNum != 10)                                //判断是否发送成功
        return 0xFF;                               //发送失败返回 0xFF
```

数据接收，以接收 10B 的数据为例：

```
unsigned char RxData[10];
    DWORD RxNum;
    ReadFile(hPort,RxData,10,&RxNum,NULL);
    if(RxNum != 10)                                //判断是否接收成功
    {
        PurgeComm(hPort,PURGE_RXCLEAR);
        return 0xFF;                               //接收失败返回 0xFF
    }
```

程序实现了串行数据的收发，再配合单片机软件即可实现基于 USB 总线适配卡的基本通信功能。还可以根据具体需要进一步完善单片机的软件功能，便于现场应用。

6.6 以太网与 CAN 接口设计

6.6.1 以太网简介

以太网（Ethernet）最早由Xerox（施乐）公司创建，1980 年，DEC、lntel 和 Xerox 三家公司将其联合开发成为一个标准（DIX80）。1983 年，这个标准被 IEEE 采纳，发展为 IEEE802.3 标准。目前，以太网成为应用最为广泛的局域网技术，包括标准以太网(10Mbit/s)、快速以太网(100Mbit/s)和千兆以太网(1Gbit/s)等。

以太网基于网络上无线电系统多个节点发送信息的想法实现，每个节点必须取得电缆或者其他信道使用权才能传送信息，由于19 世纪的物理学家认为存在一种叫做光以太（Ether）的物质，是电磁波传播的介质，以太网因此得名。以太网每一个节点都有全球唯一的 48bit 地址也就是制造商分配给网卡的MAC 地址，以保证以太网上所有系统能互相鉴别。

带冲突检测的载波侦听多路访问 (CSMA/CD)技术是以太网技术的核心，它规定了多台电脑共享一个信道的方法，由于这个方法要比令牌环网或者主控制网要简单，所以也使得以太网很大程度上取代了其他局域网标准，如令牌环、FDDI 和 ARCNET 等。

以太网可以采用多种连接介质，包括同轴电缆、双绞线和光纤等。其中双绞线多用于从主机到集线器或交换机的连接，而光纤则主要用于交换机间的级联和交换机到路由器间的点到点链路上，同轴电缆作为早期的主要连接介质已经逐渐趋于淘汰。目前 PC 上的以太网接口

都是采用双绞线。

与 USB 端口等其他接口相比，以太网具有传输距离远、组网方便、可以通过无线路由器与智能手机和平板电脑等 WIFI 设备通信的诸多优点；并且各种主流操作系统均自带网卡驱动程序和网络协议栈，用户不必自行编写驱动程序，只需采用操作系统提供的网络套接字进行编程即可，开发周期短。因此，F-小节将介绍采用以太网接口的 CAN 接口设备的硬件和软件设计方法。

6.6.2 硬件电路设计说明

随着芯片集成度越来越高，现在的微处理器外设也越来越丰富。为了方便开发，这里选择意法半导体公司基于 ARM®Cortex™-M3 内核的互联型微处理器 STM32F107 作为主控芯片，STM32F107 同时具有两个 CAN 总线接口和一个 10M/100M 自适应以太网接口，最高时钟频率可达 72MHz，并提供 256KB 的 Flash 存储空间和 64KB 的 SRAM 空间。

以太网 CAN 接口设备的硬件电路主要由 4 部分组成，分别为 CPU 及外围电路、以太网接口电路、CAN 总线接口电路及电源电路。STM32F107 内部已经集成了存储器，因此 CPU 部分的电路比较简单，如图 6-34 所示，只包含晶体振荡电路和仿真器接口电路。

图 6-34 STM32F107 CPU 及外围电路

STM32F107 中的以太网模块只提供以太网的 MAC 层，不包含物理层，因此需要外扩以太网物理层芯片。STM32F107 可以与采用标准 MII 或者 RMII 接口的以太网物理层芯片相连，本电路选择 DP83848 作为物理层芯片，如图 6-35 所示，图中的 CN3 是集成了网络变压器的 RJ45 插座。



图 6-35　以太网接口电路

同 SJA1000 一样，STM32F107 的 CAN 总线模块也需要外接总线驱动芯片才能正常工作。图 6-36 是 CAN 总线接口电路的电路图，为了避免 CAN 总线可能对系统带来的干扰和损害，采用总线隔离芯片 HCPL-0930 进行了总线隔离。

图 6-36　CAN 总线接口电路

图 6-37 是电源部分的电路图。系统由 5V 电源供电，供电电源经过线性调压芯片 U12

图 6-37　电源电路

后变为 3.3V 电压，为 CPU 和以太网部分供电；U11 是一个 DC/DC 隔离电源，用来给 CAN 总线部分供电，以实现总线隔离。

6.6.3　以太网接口卡软件设计

以太网 CAN 接口设备的软件运行于开源操作系统 FreeRTOS 上，主要由以太网接口、网络协议栈、CAN 总线收发以及数据转发 4 个部分组成，均采用 C 语言编写。

以太网接口部分的主要功能是完成以太网数据帧的收发。STM32F107 的以太网功能强大，支持 DMA 功能，当有以太网数据帧到来时，能够自动判断该帧是否有效，将有效帧存入相应的缓冲区，并通知用户程序处理；用户程序需要发送数据时，只要将数据帧写入空闲的缓冲区地址，STM32F107 就会在网络空闲时将其发送出去。意法半导体公司提供了以太网接口部分的软件库，用户只需要调用这个软件库，就可以实现对以太网部分的全部操作，因此，以太网接口部分的软件可以参考意法半导体公司的软件库使用手册。

STM32F107 只提供网络的 MAC 层，需要网络协议栈软件来实现 ISO/OSI 参考模型中的其他几层功能。网络协议栈是一套比较复杂的软件系统，目前有一些适用于小型嵌入式系统的开源轻型协议栈可供利用，如 μIP 和 LwIP 等，采用这些开源协议栈可以大幅降低网络软件的开发难度。

本系统选用 LwIP 网络协议栈，LwIP 全称是 Light Weight（轻型）IP 协议，有无操作系统的支持都可以运行。LwIP 实现的重点是在保持 TCP/IP 协议主要功能的基础上减少对 RAM 的占用，一般它只需要几百字节的 RAM 和 40KB 左右的 ROM 就可以运行，这使 LwIP 协议栈非常适合在低端的嵌入式系统中使用。

使用 LwIP 协议栈，还要做一些移植工作，主要是协议栈与 MAC 层的接口部分，需要提供 low_level_output 和 low_level_input 两个底层输入输出函数。

```
static struct pbuf *low_level_input( struct netif *netif )
{
        struct pbuf *p = NULL;
        struct pbuf *q;
        static xSemaphoreHandle xRxSemaphore = NULL;

        u16_t len;
        u32_t tmp;
        int l =0;
        FrameTypeDef frame;
        u8 *buffer;
taskENTER_CRITICAL( );
        //等待以太网数据帧接收完毕
        if (ETH_Rx_IsReady( ) == (u32)SET){
            ( void ) netif;
            if( xRxSemaphore == NULL ){
                vSemaphoreCreateBinary( xRxSemaphore );
            }
             //使用信号量互斥来访问 MAC 硬件层
```

```
                if( xSemaphoreTake( xRxSemaphore, netifGUARD_BLOCK_TIME ) ){
                    frame = ETH_RxPkt_ChainMode( );                        //读取数据帧
                    len = frame.length;
                    buffer = (u8 *)frame.buffer;
                    p = pbuf_alloc(PBUF_RAW, len, PBUF_POOL);              //根据帧长度从 pbuf 池中
                                                                           //分配一个 pbuf 链
                    if (p != NULL){
                        for (q = p; q != NULL; q = q->next){
                            memcpy((u8_t*)q->payload, (u8_t*)&buffer[l], q->len);
                            l = l + q->len;
                        }
                    }
                    frame.descriptor->Status = ETH_DMARxDesc_OWN;          //设置接收描述符中的所
                                                                           //有者位
                    ETH_Rx_DMAResume( );
                    xSemaphoreGive( xRxSemaphore );                        //释放信号量
                }
            }
            taskEXIT_CRITICAL( );
            return p;
    }

    static err_t low_level_output( struct netif *netif, struct pbuf *p )
    {
            static xSemaphoreHandle xTxSemaphore = NULL;
            struct pbuf *q;
            err_t xReturn = ERR_OK;
            int l = 0;
            u8 *buffer =   (u8 *)ETH_GetCurrentTxBuffer( );
            ( void ) netif;
            if( xTxSemaphore == = NULL ){
                vSemaphoreCreateBinary( xTxSemaphore );
            }
            //使用信号量互斥来访问 MAC 硬件层
            if( xSemaphoreTake( xTxSemaphore, netifGUARD_BLOCK_TIME ) ){
                  for(q = p; q != NULL; q = q->next) {
                    memcpy((u8_t*)&buffer[l], q->payload, q->len);         //将需要发送的数据复制
                                                                           //到缓冲区
                    l = l + q->len;
                  }
                ETH_TxPkt_ChainMode(l);
                xSemaphoreGive( xTxSemaphore );
                }
                return xReturn;
    }
```

CAN 总线收发部分主要提供 CAN 总线数据收发函数，这部分在前面 5.3.5 节已经进行了

介绍，在此不再重复。

数据转发部分是将接收到的以太网数据经过处理后转发到 CAN 总线上，同时将接收到的 CAN 总线数据打包转发到以太网上。以太网到 CAN 的转发和 CAN 到以太网的转发分别由两个操作系统任务（Task）完成，这两个任务由操作系统在合适的时候调度。以下是这两个任务的代码。

```
//以太网到 CAN 的转发任务
static portTASK_FUNCTION( vLanToCanTask, pvParameters )
{
        //变量声明
        struct netconn    *CanTxConn;                //LwIP  所需的网络连接指针
        struct netbuf    *CanTxNetBuf;               //LwIP  所需的网络缓冲区指针
        struct ip_addr *RemoteIpAddr;                //主机 IP 地址
        struct Can_Txframe Can_Txframe;              //CAN 数据帧结构体
        struct LanToCanPar * LanToCanPar;            //任务初始化参数结构体
        CanTxMsg TxMessage;                          //用于 CAN 收发的数据结构，参考 STM32 标准库
        u8 TransmitMailbox1;                         //CAN 邮箱号
        u16 i, j;
        u16 charrxed;
        u8 *data;
        u8 datap;
        u8 tempdata[9];
        //获得初始化参数，主要是本地接收端口号
        LanToCanPar = ((struct LanToCanPar *)(pvParameters));
        //建立一个 UDP 连接，用于接收主机需要发送的 CAN 数据
        CanTxConn = netconn_new(NETCONN_UDP);
        netconn_bind(CanTxConn, NULL, LanToCanPar -> port);

        CanTxNetBuf = netbuf_new( );                 //分配网络缓冲区

        datap = 0;
         TxMessage.RTR=CAN_RTR_DATA;                 //设置帧类型为数据帧
         TxMessage.IDE=CAN_ID_STD;                   //使用标准帧
         TxMessage.DLC=8;                            //数据长度为 8B

         while(1){
           //判断是否收到 UDP 数据包
             while((CanTxNetBuf = netconn_recv(CanTxConn)) != NULL ){
                 do{
                       //读取 UDP 数据包到 data 数组中，charrxed 为数据长度
                       netbuf_data(CanTxNetBuf, &data, &charrxed);
                       /*数据长度是 10 的倍数，每 10B 是一个需要发送的 CAN 数据包，其中前
                       两个字节为 CAN 帧地址*/
                       for (i =0; i < charrxed / 10; i++){
                             //读取 CAN 帧地址，写入到 CAN 发送数据结构中
                             TxMessage.StdId = (data[datap] | data[datap+1] << 8);
```

```
                    datap += 2;
                    //读取 8B 的 CAN 数据写入 CAN 发送数据结构中
                    for (j=0; j<8; j++){
                        TxMessage.Data[j]=data[datap++];
                    }
                    //执行 CAN 数据帧发送，函数返回值为此次发送所使用的邮箱号
                    TransmitMailbox1=CAN_Transmit(CAN1,&TxMessage);
                    //等待 CAN 数据帧发送完毕
                    While (CAN_TransmitStatus(CAN1,TransmitMailbox1) != CANTXOK);
                }
                datap = 0;
            } while(netbuf_next(CanTxNetBuf) >= 0);
            netbuf_delete(CanTxNetBuf);          //释放网络缓冲区
        }
    }
}

//CAN 到以太网的转发任务
static portTASK_FUNCTION( vCanToLanTask, pvParameters )
{
    //变量声明
     struct netconn   *CanRxConn;                    //LwIP 所需的网络连接指针
     struct netbuf    *CanRxNetBuf, *CanPortNetBuf;  //LwIP 所需的网络缓冲区指针
     struct Can_Rxedframe Can_Rxedframe;            //CAN 数据帧结构体
     struct CanToLanPar *CanToLanPar;               //任务初始化参数结构体
     u16 RemotePort;                                //主机端口号
     CanRxMsg RxMessage;                            //用于 CAN 收发的数据结构，参考 STM32
                                                    //标准库
     char data[1000];                               //用于网络数据暂存的缓冲区
     int i;
     u16 chartotx = 0;                              //需要发送的数据字节计数
     //获得初始化参数，主要是主机 IP 地址和端口号
     CanToLanPar = ((struct CanToLanPar *)(pvParameters));
     //建立一个 UDP 连接，用于向主机发送接收到的 CAN 数据
     CanRxConn = netconn_new(NETCONN_UDP);
     netconn_connect(CanRxConn, CanToLanPar -> ipaddr, CanToLanPar -> port);
     //分配网络传输缓冲区
     CanRxNetBuf = netbuf_new( );

     while(1){
        //判断是否收到 CAN 总线数据帧
        while( xQueueReceive( CanToLanPar->xRxedQueueHandle, &Can_Rxedframe, 0 ) ){
            //如果收到了 CAN 总线数据帧，首先读出 CAN 数据帧地址，写入到缓冲区
            data[chartotx ++] = Can_Rxedframe.CanTxMsg.StdId & 0xff;
            data[chartotx ++] = (Can_Rxedframe.CanTxMsg.StdId >> 8) & 0xff;
            //再将 8B 的数据写入缓冲区
```

```
			for (i=0; i<8; i++){
				data[chartotx + i] = Can_Rxedframe.CanTxMsg.Data[i];
			}
			chartotx += 8;					//修正字节计数
			if (chartotx >= 1000)			//如果字节数大于 1000，跳出循环
				break;
		}
		//如果发送字节数大于零，则执行一次网络数据包发送
		if (chartotx > 0){
			netbuf_ref(CanRxNetBuf, data, chartotx);
			netconn_send(CanRxConn, CanRxNetBuf);
			chartotx = 0;
		}
		vTaskDelay(1);					//调用 FreeRTOS 的任务延迟函数，将任务延迟 1ms
	}
}
```

6.7 本章小结

至此介绍了 6 种不同的 CAN 总线通信节点的设计，这些都是基于工程应用的实践需要开发研制的。包括连接 PC 所需的 PC-104、ISA、并口、PCI、USB、以太网接口卡，实现了一系列 CAN 总线适配卡的设计研发，在工程应用中均取得了较理想的效果。尤其是基于 USB 总线的 CAN 适配卡，更是具有结构简单、取电方便、接口简洁等优点。

思考题与习题

6.1 简述 ISA 总线的 I/O 读写时序以及与 SJA1000 接口的解决方案。
6.2 简述 PCI 总线的主要特点。
6.3 简述 PC 并行端口 EPP 工作模式的主要功能。
6.4 简述“USB-串口”转换芯片 FT232BM 的主要功能。
6.5 与 USB 等 PC 总线相比，以太网的主要优点是什么？

第 7 章　CAN 总线的工程应用问题

前面章节介绍了 CAN 总线协议和实现 CAN 总线协议的 CAN 总线控制器 SJA1000 的基本使用。本章主要对扩展模式下 SJA1000 的应用问题进行了详细的论述；对 SJA1000 的滤波器配置问题进行了分析；对 CAN 驱动器的应用问题进行了介绍；最后，分析了网络时延的来源以及 CAN 总线实时性能的提升问题。

本章要点

- SJA1000 扩展模式下的功能分析；
- SJA1000 的滤波器配置问题；
- PCA82C250/251 总线驱动器的应用问题；
- CAN 总线的实时性分析。

7.1　CAN 总线的扩展模式功能分析

7.1.1　寄存器和 RAM 地址分配

SJA1000 寄存器和报文缓冲器对于主控制器来说是外围寄存器，它们可以通过复用的地址/数据总线寻址。在不同的模式（操作或复位）可以访问不同的寄存器。正常操作的地址范围是：Address 0～31。它包括用于初始化、状态和控制的寄存器。而且 CAN 报文存储器位于地址 16 和 28 之间。在主控制器写访问时，用户能够寻址 CAN 控制器的发送缓冲器，在读访问时，读出接收缓冲器的内容。

除了上面所说的范围外，整个接收 FIFO 映射在 CAN 地址 32 和 95 之间（见图 7−1）。而且，是内部 80B RAM 的一部分，SJA1000 发送缓冲器在 CAN 地址 96 和 108 之间。图 7−1 显示了物理 RAM 地址和 CAN 地址之间的关系。

直接访问 RAM 时，可以读发送缓冲器和完整的接收 FIFO。

PeliCAN 模式里，接收 FIFO 能够存储高达 n=21 条报文。用下面的等式，可以计算出报文的最大数量：

$$n = \frac{64}{3 + \text{data_length_code}}$$

接收缓冲器被定义为一个 13B 的窗口，总是包括接收 FIFO 当前的接收报文。见第 3 章的图 3−4，CAN 的部分或全部的报文都在接收缓冲器窗口，但是，在“释放接收缓冲器”命令之前，接收 FIFO 里的下一个接收到的报文在接收缓冲器窗口（从 CAN 地址 16 开始）将完全可以看到。

为了分析的需要，SJA1000 另外提供两个寄存器，处理接收报文：

① RX 缓冲器起始地址寄存器（RBSA）允许接收 FIFO 范围里识别单个 CAN 的报文；

② RX 计数器寄存器，表示接收 FIFO 里的当前存储的报文数量。

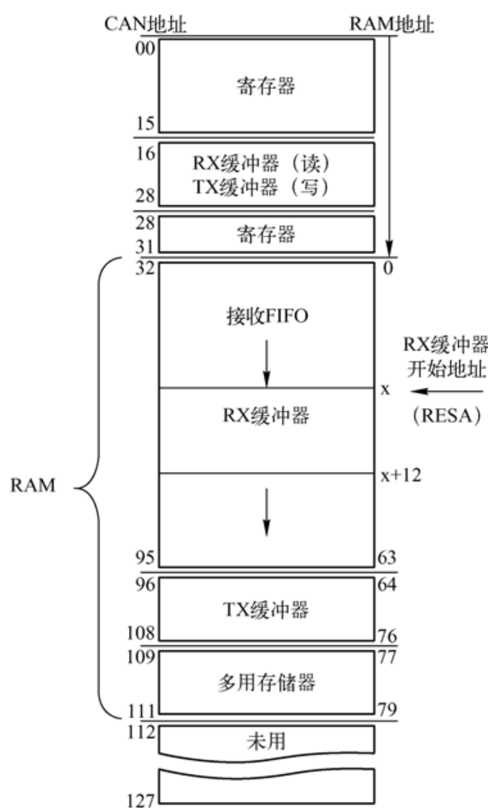


图 7-1　寄存器和 RAM 地址分配

7.1.2　错误处理功能

基于错误计数器的值，每个 CAN 控制器能够在三种错误状态之一中工作：错误激活、错误认可或总线离线。如果错误计数器的值在 0～127 之间，CAN 控制器是错误激活的。此时产生错误激活标志（6 个隐性位）。如果发送错误计数器的值高于 127，则到达总线离线状态。在这种状态下，自动置位复位请求位，SJA1000 对总线没有影响。如图 7-2 所示，总线离线状态只能在主控制器用命令“Reset Request=0”退出。这将启动总线离线恢复定时器，发送错误计数器计数 128 个总线释放信号。计数结束后，两个错误计数器都是 0，器件再次处于错误激活状态。

1．出错计数器

如上面描述，CAN 的错误状态与发送错误计数器和接收错误计数器的值直接有关。为了仔细研究错误界定，支持 SJA1000 的增强错误分析功能，CAN 控制器提供可读的错误计数器。另外，在复位模式，允许对于两个错误计数器进行写访问。

2．出错中断

如图 7-2 所示，使用了 3 个错误中断源来向主控制器发送出错的状态。每个中断都能在中断使能寄存器里分别使能。

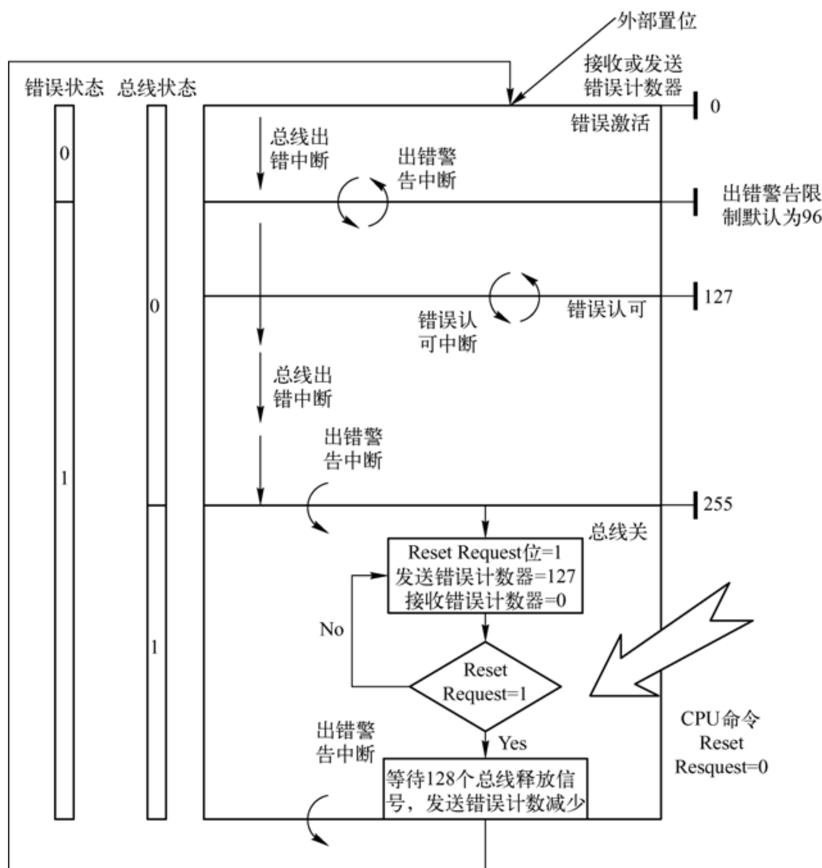


图 7-2　SJA1000 的出错中断

（1）总线出错中断

在 CAN 总线上检测到任何一个错误都会产生中断。

（2）出错警告中断

如果超过出错警告界限，产生出错警告中断。而且它在 CAN 控制器进入总线离线状态和在此之前再一次进入错误激活状态也会产生这个中断。SJA1000 的出错警告界限在复位模式中可编程。复位后的默认值是 96。

（3）错误认可中断

如果错误状态从错误激活变成错误认可或相反，将产生错误认可中断。

3．错误代码捕获功能

SJA1000 可以执行在 CAN2.0B 规范中定义的所有错误界定。每个 CAN 控制器处理错误的整个过程是完全自动的，如图 7-3 所示。

图 7-3 中，为了向用户提供某个错误的详细信息，SJA1000 提供了错误代码捕捉功能。无论什么时候发生 CAN 总线错误，它都会强制产生相应的总线出错中断。同时，当前位的位置被捕捉入错误代码捕捉寄存器。在主控制器将捕捉的数据读出前，它都会被保存在寄存器中。然后捕捉机制再次激活。寄存器可以根据内容区分四种错误类型：格式出错、填充出错、

位出错和其他错误。

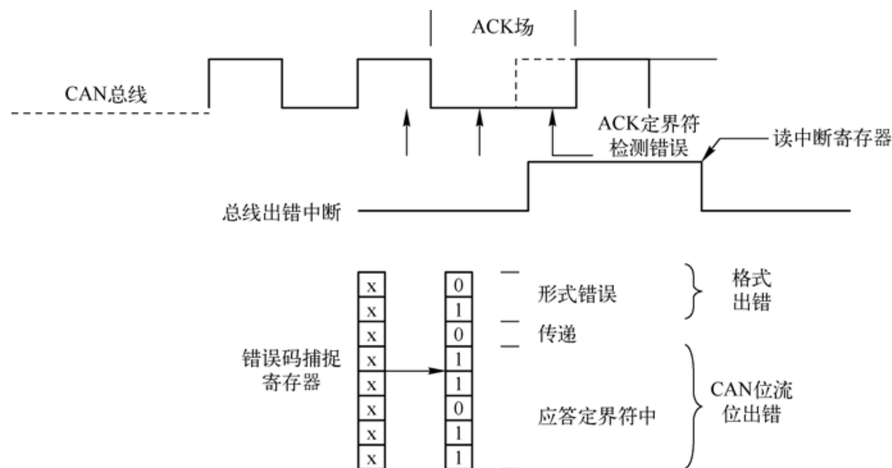


图 7-3　错误码捕捉功能举例

根据 CAN 规范定义，CAN 总线上的每个位只有特殊类型的错误。表 7-1 和表 7-2 显示了在 CAN 报文发送和接收期间可能出现的所有错误。左边的部分包括位置和错误的类型，这些由错误码捕捉寄存器捕捉。每张表的右边部分是将错误码转换成上层的错误描述，可直接从寄存器内容知道其含义。通过使用这些表格，能得到有关错误计数器的变化和器件发送和接收管脚的错误状态的更多信息。使用这些表时，例如在错误分析软件里，可以详细地分析每一个错误状态。关于 CAN 错误类型和位置的信息能用于错误统计和系统维护或在系统优化期间进行纠正。

表 7-1　接收时可能出现的错误

错误码捕捉		RX 错误计数	描　　述	
CAN 位流里的错误位置	错误类型			
标识符 SRR、IDE 和 RTR 位，保留位、数据长度码、数据场、CRC 序列。	填充	+1	收到 5 个电平相同的连续的位	——
CRC 定界符	格式填充	+1	RX=显性，收到超过 5 个电平相同的连续位。	位必须是隐性
应答符	位	+1	TX=显性，但 RX=隐性。	不能写显性位
应答定界符	格式	+1	RX=显性，或检测到 CRC 错误。	临界的总线定时或总线长度，CRC 序列不正确。
帧结束	格式 其他	+1 ±0	RX=头六个位是显性位 RX=最后一位是显性位	—— 发出超载标志，如发送器重新发送，数据可能重复
间隔	其他	±0	RX=显性	接收器发出超载标志
激活错误标志	位	+8	TX=显性，但 RX=隐性	不能写显性位
容许的显性位（Tolerate Dominant Bit）	其他	+8	RX=出错标志后的第一位是显性 RX=出错或过载标志后有超过 7bit 显性位	
错误定界符	格式 其他	+1 ±0	RX=头七位是显性位 RX=定界符的最后一位是显性位	—— 发送超载标志
超载标志	位	+8	TX=显性，但 RX=隐性	不能写显性位

表 7-2　发送时可能出现的错误

错误码捕捉		RX 错误计数	描　　述	
CAN 位流里的错误位置	错误类型			
帧起始	位	+8	TX=显性，但 RX=隐性	不能写显性位
标识符	位 填充	+8 ±0	TX=显性，但 RX=隐性 TX=隐性，但 RX=显性	不能写显性位 ——
SRR 位	位 填充	+8 ±0	TX=显性，但 RX=隐性 TX=隐性，但 RX=显性	不能写显性位 ——
IDE 和 RTR 位	位 填充	+8 +8	TX=显性，但 RX=隐性 TX=隐性，但 RX=显性	不能写显性位 ——
保留位、数据长度码、数据场、CRC 序列	位	+8	TX=显性，但 RX=隐性	不能写显性位
CRC 定界符	格式	+8	RX=显性	位必须为隐性
应答隙	其他 其他	+8 ±0	RX=隐性（错误激活） RX=隐性（错误认可）	没有应答 没有应答，节点可能单独在总线
应答定界符	格式	+8	RX=显性	临界的总线定时或总线长度
帧结束	格式其他	+8 +8	RX=头六个位是显性位 RX=最后一位是显性位	—— 帧已经被一些节点接收，再次发送可能导致接收器里数据重复
间隔	其他	±0	RX=显性	来自于“旧”CAN 控制器的超载标志
激活错误标志 过载标志	位	+8	TX=显性，但 RX=隐性	不能写显性位
允许显性位（Tolerate Dominant Bit）	格式	+8	RX=在激活错误标志或过载标志后有超过 7 个显性位	——
错误定界符	格式其他	+8 ±0	RX=头 7bit 是显性位 RX=界定符的最后一位是显性位	—— 来自于“旧”CAN 控制器的超载标志
认可错误标志	其他	+8	RX=显性（错误认可）	没有收到应答，节点不是单独在总线上

7.1.3　仲裁丢失捕捉功能

SJA1000 能够确定 CAN 位流丢失仲裁的确切位置，并立即产生“仲裁丢失中断”。而且，这个位的号码被捕捉到仲裁丢失捕捉寄存器。主控制器读出这个寄存器的内容后，仲裁丢失捕捉功能被再次激活。

在这个功能的帮助下，SJA1000 能够监控每个 CAN 总线的访问。诊断或在系统配置期间，可以识别仲裁失败的所有位置。

7.1.4　单次发送功能

在一些应用中，自动重发 CAN 报文没有意义：它造成一个节点几次仲裁和数据变得无效。

为了请求一个“单次发送”，CAN 控制器必须完成下面的步骤：

① 发送请求；

② 等待发送状态；

③ 中止发送。

通过同时置位命令 CMR.0 和 CMR.1，处理软件能将初始化“单次发送”选项减少到一个

命令。

在这种情况下，没有必要查询状态位，主控制器能集中于其他任务上。“单次发送”功能与 SJA1000 的仲裁丢失和错误代码捕捉功能完美结合。

如果仲裁丢失或发生错误，SJA1000 不会重新发送报文。一旦置位状态寄存器的发送状态位，内部发送请求就自动清除。

使用两个捕捉寄存器的附加信息，一个报文是否被重发由用户控制。

如在前面章节里描述的，单次发送能和自我测试模式一起使用。

7.1.5 自动位速率检测功能

SJA1000 支持 PeliCAN 模式的自动位速率检测的请求。这里将简短地描述一个不影响网络运行操作的应用例子。在仅听模式，SJA1000 不能发送报文也不能产生出错帧。这个模式里只能接收报文。软件里预定义的表格包含所有可能的位速率以及它们的位定时参数。在启动用最高位速率接收报文之前，SJA1000 使能接收中断和溢出中断。如果在 CAN 总线产生了错误，软件会转向下一个较低的位速率。在成功地接收到一个报文后，SJA1000 已经检测到正确的位速率而且能转向正常工作模式。

7.1.6 仅听模式

在仅听模式里，SJA1000 不能在 CAN 总线上写显性位。激活错误标志或超载标志也不能都写，成功接收后的应答信息也不会给出。错误就像在错误认可模式里处理一样。错误分析功能如：错误码捕捉和出错中断就像在正常操作模式一样工作。但是，错误计数器的状态被冻结。可以接收报文，但不可以发送。因此，这个模式可用于自动的位速率检测。在进入仅听模式之前，必须进入复位模式。

7.1.7 CAN 的自测试

SJA1000 支持两种不同的自测试：局部自测试和总体自测试。

对于局部自测试，例如能很好地用于单个节点测试，因为它不需要来自于其他节点的应答。此时，SJA1000 必须处于“自测模式”（模式寄存器）并发出“自接收请求”命令。

对于总体自测试，在操作模式下 SJA1000 执行同样的命令。但是，在运行系统中，需要 CAN 的应答。

注意：在这两种情况下，物理层接口包括带有终端的 CAN 总线必须有效。发送或自接收通过置位命令寄存器的相应位初始化。SJA1000 提供三个命令位用于 CAN 发送和自接收初始化。表 7-3 显示了所有可能的组合（取决于所选的工作模式）。

表 7-3　CAN 发送请求命令

命令	CMR	成功操作后的中断	自我测试模式	操作模式
自接收请求	0x10	RX 和 TX	局部自测试	总体自测试
发送请求	0x01	TX	正常发送	正常发送
单次发送	0x03	TX	没有重发的发送	没有重发的发送
单次发送和自我接收请求	0x12	RX 和 TX	无重发的局部自测试	无重发的总体自测试

7.2 CAN 总线的滤波器配置问题

独立的 CAN 控制器 SJA1000 装配了一个多功能的验收滤波器，该滤波器允许自动检查标识符和数据字节。使用这些有效的方法，可以防止对于某个节点无效的报文或报文组存储在接收缓冲器里。因此降低了主控制器的处理负载。

滤波器由验收码寄存器和屏蔽寄存器根据下面给定的算法来控制。接收到的数据会和验收代码寄存器中的值进行逐位比较。接收屏蔽寄存器定义与比较相关的位的位置（0=相关，1=不相关）。只有收到报文的相应的位与验收代码寄存器相应的位相同，报文才会被接收。

7.2.1 BasicCAN 模式的验收滤波

SJA1000 在这个模式可以直接取代 PCA82C200（硬件和软件）。因此验收滤波功能与 PCA82C200 的一样，也可以使用。这个滤波器是由两个 8bit 寄存器——验收代码寄存器（ACR）和验收屏蔽寄存器（AMR）控制。CAN 报文标识符的高 8bit 和这些寄存器里的值相比较，如图 7-4 所示。因此任何一个节点可以自定义标识符并接收相应的数据。

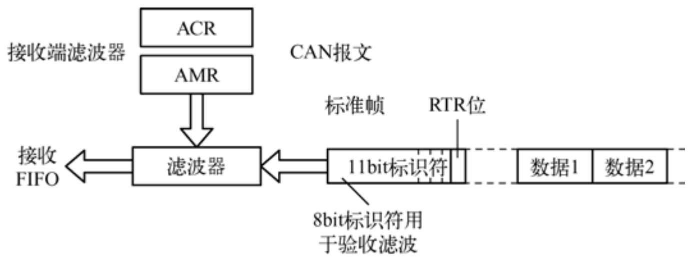


图 7-4 BasicCAN 模式的验收滤波

在验收屏蔽寄存器里是“1”的位置上，标识符相应的位可以是任何值。这对于三个最低位也一样。因此在如图 7-5 所示的这个例子里可以接收 64 个不同的标识符。标识符其他的位必须等于验收代码寄存器相应位的值。

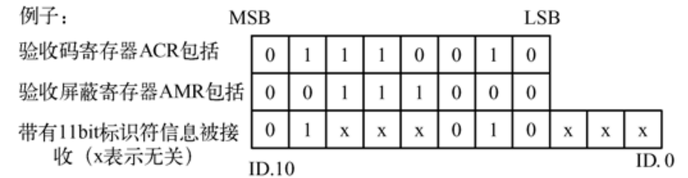


图 7-5 验收滤波举例

7.2.2 PeliCAN 模式的验收滤波

PeliCAN 模式的验收滤波已被扩展：4 个 8bit 的验收码寄存器（ACR0，ACR1，ACR2 和 ACR3）和验收屏蔽寄存器（AMR0，AMR1，AMR2 和 AMR3）可以用多种方法滤波报文。如图 7-6 和图 7-7 所示。

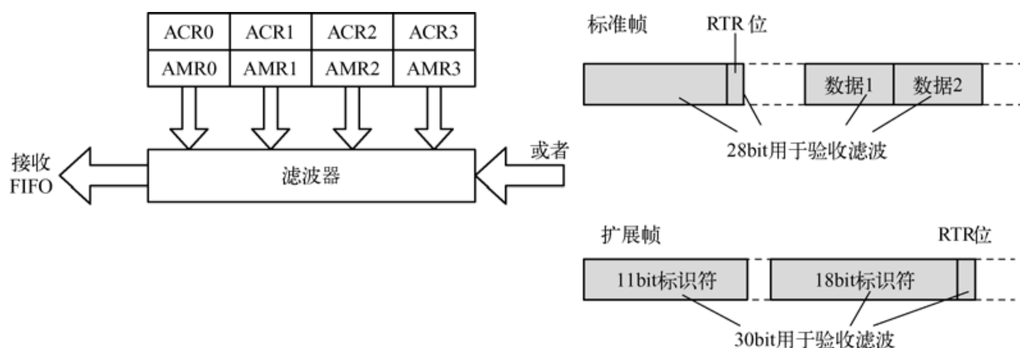


图 7-6　PeliCAN 模式的验收滤波（单滤波器模式）

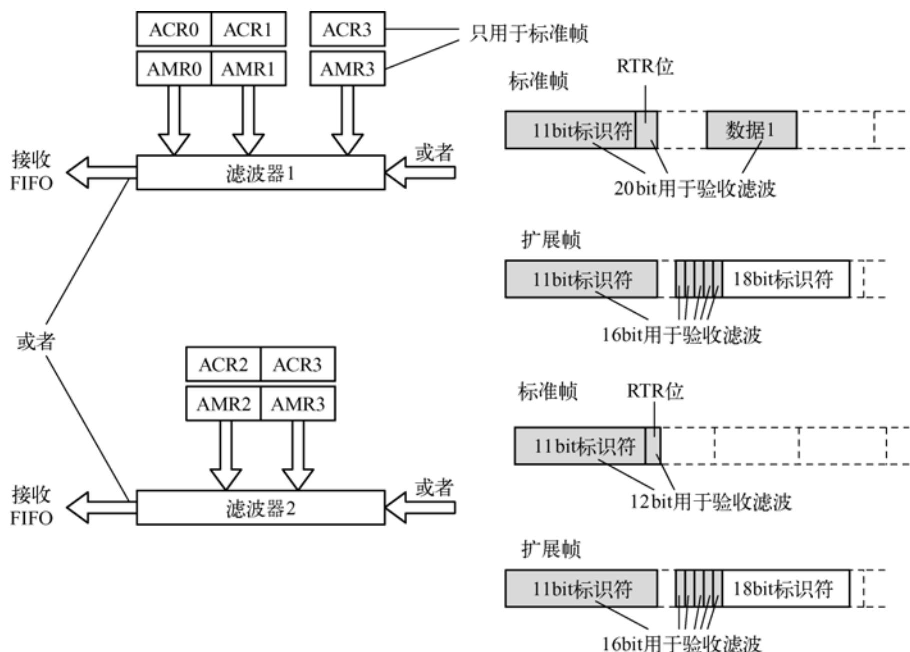


图 7-7　PeliCAN 模式的验收滤波（双滤波器模式）

上图中的这些寄存器可用于控制一个长的滤波器或两个短的滤波器。报文的哪些位用于验收滤波，取决于收到的帧（标准帧或扩展帧）和选择的滤波器模式（单滤波器或双滤波器）。有关报文验收滤波的更多信息见表 7-4。从图和表可以看出，标准帧的验收滤波器可以包括 RTR 位甚至数据字节。对于不需要经过验收滤波的报文位（例如报文组被定义为接收）验收屏蔽寄存器必须置位相应的位。

表 7-4　PeliCAN 模式的验收滤波器总结

帧类型	单滤波器模式	双滤波器模式
标准	验收的报文位： —11bit 标识符 —RTR 位 —第一个数据字节（8bit） —第二个数据字节（8bit）	滤波器 1 验收的报文位： —11 位标识符、RTR 位、第一个数据字节（8bit） 使用的验收码寄存器和屏蔽寄存器： —ACR0/ACR1 和 ACR3 的低四位

（续）

帧类型	单滤波器模式	双滤波器模式
标准	使用的验收码寄存器和屏蔽寄存器： —ACR0 和 ACR1/ACR2/ACR3 的高四位 —AMR0 和 AMR1/AMR2/AMR3 的高四位 （接收屏蔽寄存器的未使用的位应设为“1”）	—AMR0/AMR1 和 AMR3 的低四位 滤波器 2 用于验收测试的报文位：11bit 标识符、RTR 位 使用的验收码寄存器和验收屏蔽寄存器： —ACR2 和 ACR3 的高四位 —AMR2 和 AMR3 的高四位
扩展	用于验收的报文位： —11bit 基本的标识符 —18bit 扩展的标识符 —RTR 位 使用的验收码和验收屏蔽寄存器： —ACR0/ACR1/ACR2 和 ACR3 的高六位 —AMR0/AMR1/AMR2 和 AMR3 的高六位 （验收屏蔽寄存器的未使用的位应设为“1”）	滤波器 1 用于验收的报文位：11bit 基本标识符、扩展标识符的 5 个最高位 使用的验收码和验收屏蔽寄存器： —ACR0/ACR1 和 AMR0/AMR1 滤波器 2 用于测试验收的报文位：11bit 基本的标识符、扩展标识符的 5 个最高位 使用的验收码和验收屏蔽寄存器： —ACR2/ACR3 和 AMR2/AMR3

如果报文不包括数据字节（例如：是一个远程帧或者数据长度码为零）但是验收滤波包括数据字节，则如果标识符直到 RTR 位都有效的话，报文会被接收。

7.3　CAN 总线的驱动问题

本节将针对具有代表性的 PCA82C250/251 驱动器的工程应用问题进行分析。包括斜率控制模式的介绍、最大总线线路长度的计算和总线终端拓扑结构的分析。

7.3.1　CAN 接口的斜率控制功能

PCA82C250 / PCA82C251 三种控制模式及转换在 4.2 节已有详细描述，三种模式的应用情况的比较见表 7−5。

表 7−5　三种模式说明

模式类型	工作模式所提供的特征	引脚 R_s 控制电平
高速模式	发送、接收功能，速度最高	低/悬空
准备模式	减少电流、远程唤醒	高
斜率控制模式	可变斜率、速度可调	R_{ext} 取值 10～140kΩ之间

由于斜率控制模式主要应用于非屏蔽的总线电缆，速度可以通过 R_{ext} 进行控制，应用较为普遍。所以现主要就斜率控制模式进行介绍。

在斜率控制模式中，由于非屏蔽总线电缆的应用，考虑到电磁兼容等问题，总线转换速率需要被降低，以下进行转换速率的计算。

在斜率控制模式中，总线的转换速度可以通过连接在控制引脚 $\mathrm{R_s}$ 上的串联阻抗 R_{ext} 来进行调整。总线输出信号的转换速率 SR 和流出引脚 $\mathrm{R_s}$ 的电流 I_{Rs} 成比例。由于电流主要由斜率控制电阻的阻值 R_{ext} 决定，所以使用不同阻值的 R_{ext} 就有不同的转换速度。图 7−8 是芯片供应商给出的典型的单端转换速度值 SR 和斜率控制阻抗值 R_{ext} 之间的关系。

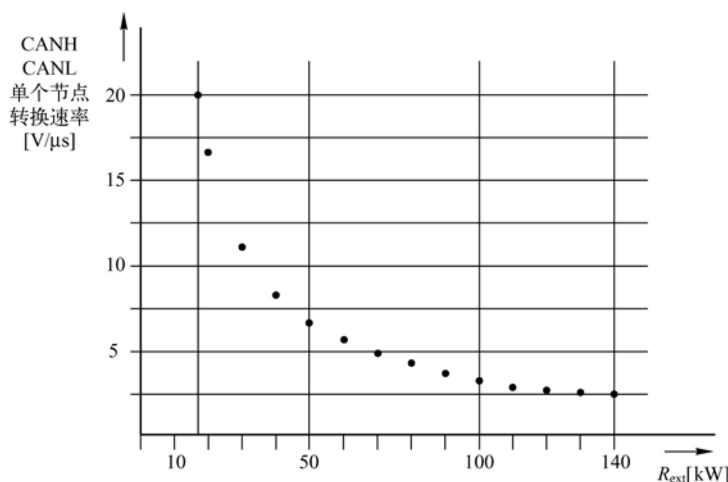


图 7-8　转换速度和斜率控制阻抗值的关系

由上图可以看出，随着串联阻抗 R_{ext} 的增大，总线输出信号的转换速率 SR 逐渐变小，下面求取转换速率 SR 与串联阻抗 R_{ext} 的关系式。

首先求取单端转换速度常数 K_{SE}：

单端转换速度常数 K_{SE} 有关系式：$I_{SR}=\dfrac{V_{Rs}}{R_{ext}}=K_{SE}SR$

利用上图中一组典型数据：

转换速率 SR=7V/μs，串联阻抗 R_{ext}=47kΩ，R_s 的电压 $V_{Rs}=0.5\times V_{CC}$。

可以得到 K_{SE}：

$$K_{SE}=\frac{0.5V_{CC}}{R_{ext}SR}=\frac{2.5\text{V}}{47\text{k}\Omega\times 7\dfrac{\text{V}}{\mu\text{s}}}=7.6\times 10^{-3}\mu\text{s/k}\Omega$$

但在实际斜率控制模式中，电阻 R_{ext} 通常连接在 R_s 输入和由端口输出或地线提供的逻辑低电平 V_{OL} 之间。因此，单端转换速度 SR 和阻抗 R_{ext} 之间的关系由下面的等式给出：

$$SR=\frac{V_{Rs}-V_{OL}}{K_{SE}R_{ext}}=\frac{V_{Rs}-V_{OL}}{7.6\times 10^{-3}\dfrac{\mu\text{s}}{\text{k}\Omega}\times R_{ext}} \tag{7-1}$$

由此得出了转换速率 SR 与串联阻抗 R_{ext} 的关系式。

下面举两个实例，分别计算转化速率和斜率控制电阻：

例 1：已知 V_{CC}=5V，R_{ext}=36kΩ，V_{OL}=0V，则计算可得系统单端转换速度 SR 是：

$$SR=\frac{V_{Rs}-V_{OL}}{K_{SE}R_{ext}}=\frac{0.5V_{CC}}{7.6\times 10^{-3}\dfrac{\mu\text{s}}{\text{k}\Omega}\times R_{ext}}=\frac{0.5\times 5\text{V}}{7.6\times 10^{-3}\dfrac{\mu\text{s}}{\text{k}\Omega}\times 36\text{k}\Omega}=9.14\text{V}/\mu\text{s}$$

例 2：已知 V_{CC}=5V，$SR=8\dfrac{\text{V}}{\mu\text{s}}$，则计算可得系统的斜率控制电阻 R_{ext} 是：

$$R_{ext}=\frac{V_{Rs}-V_{OL}}{K_{SE}\times SR}=\frac{0.5V_{CC}}{7.6\times 10^{-3}\dfrac{\mu\text{s}}{\text{k}\Omega}\times SR}=\frac{0.5\times 5\text{V}}{7.6\times 10^{-3}\dfrac{\mu\text{s}}{\text{k}\Omega}\times SR}=41.1\text{k}\Omega\Rightarrow R_{ext}=41.1\text{k}\Omega$$

7.3.2　最大总线线路长度及节点数

影响总线长度的主要因素：

1）总线节点（CAN 控制器驱动器等）的循环延迟以及总线线路的延迟。CAN 总线的应答机制，意味着成功接收到报文的节点必须在“应答间隙”期间发送一位显性位，以表示成功接收到一帧数据。比如：总线通信速率为 250kbit/s，传送一个 bit 所需时间为 1/250*1000=4μs，那么该信号在总线的延时时间必须小于 2μs 才能保证发送节点成功地在应答间隙期间接收到该显性电平。对于双绞线而言，信号在其中的传播延时时间约为 5ns/m（典型值）。当通信速率达到 1Mbit/s 时，40m 总线长度的延时时间就达到 200ns，而允许延时时间为 600ns 左右，因此必须考虑传播延时。因而，非破坏性总线仲裁和帧内应答限制了 CAN 总线速度进一步的提高。一般情况下，总线通信速率越高，通信距离越短，对物理传输线的要求就越高，所以在双绞线、屏蔽线或其他传输线的选择上，通信速率是十分关键的参数。

① 在高速模式下，PCA82C250 和 PCA82C251 最大总线电缆长度与位速率之间的关系如表 7-6 所示，其中为了获得最大的传播延迟，CAN 位定时参数可以进行优化。

表 7-6　位速度与总线长度的关系

位速度/（kbit/s）	总线长度/m
1000	30
500	100
250	250
125	500
62.5	1000

② 在斜率控制模式下的总线长度求取如下：

上述最大总线长度是在高速模式下，最大总线线路长度 L_{max} 满足：

$$L_{max}=\frac{\frac{t_{prop}}{2}-t_{loop.eff}-t_{loop.eff.oth}}{t_p} \tag{7-2}$$

式中，t_{prop} 表示最大的双向传播延迟（CAN 位定时）；$t_{loop.eff}$ 表示有效的驱动器循环延迟；$t_{loop.eff.oth}$ 表示其他元件的有效循环延迟，例如：CAN 控制器和光耦；t_p 表示总线电缆的特定延迟。

由上式可以看出，如果驱动器的循环延迟 $t_{loop.eff}$ 下降，则最大的总线电缆长度 L_{max} 将上升。由于高速模式的驱动器循环延迟比斜率控制模式的驱动器的循环延迟 $t_{loop.eff}$ 要小。所以，在高速模式下，总线电缆的最大长度可以达到更长。

在斜率控制模式中，转换速率的降低意味着总线节点循环延迟的增加。系统的最大总线长度需要相应减小。可以用下面的等式来计算减小的长度：

$$\Delta L_{max}=\frac{t_{loop.eff}(\text{斜率控制模式})-t_{loop.eff}(\text{高速模式})}{t_p}=\frac{\Delta t_{loop.eff}}{t_p} \tag{7-3}$$

假定总线电缆的特定延迟 t_p=5ns/m，斜率控制电阻 R_{ext}=47kΩ。表 7-7 给出了在此条件下，高速模式和斜率控制模式的最大总线长度差值。

表 7-7　最大总线长度的差值

	有效的循环延迟（上限是 125℃）/ns			总线长度在两种模式下的差异/m
产品	斜率控制模式	高速模式	$\Delta t_{loop.eff}$	ΔL_{max}
PCA82C250	520	155	365	~75
PCA82C251	520	155	395	~80

2）总线电缆的串联阻抗和总线节点的输入阻抗。由于总线电缆的串联阻抗和总线节点的输入阻抗会使信号幅值下降，从而影响总线长度。

节点的循环延迟以及线路的延迟在上面已经论述，下面就总线电缆阻抗的影响和节点数量进行详细分析。

1．总线电缆阻抗对总线长度的影响

为减少总线电缆上的反射，在 ISO 11898 标准中，网络的拓扑结构接近于单线，如图 7-9 所示：

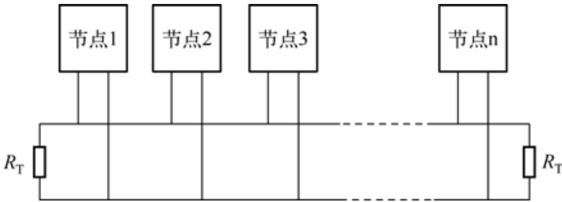


图 7-9　总线的单线拓扑结构

假设发送节点的输出电压 $V_{diff.out}$ 由一个电压源产生，发送和接收节点之间的线路阻抗用电阻 $R_ω$代表，接收节点的差动输入阻抗为 R_{diff}。则上述拓扑结构对应的等效电路如图 7-10 所示：

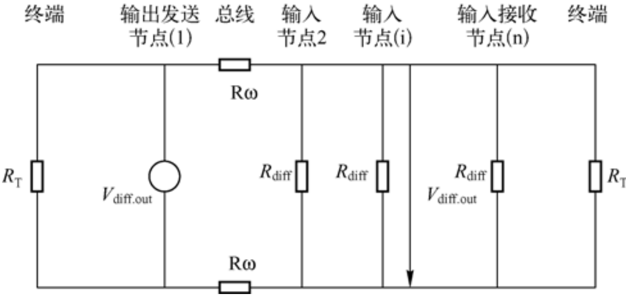


图 7-10　拓扑结构对应的等效电路

由上图可以看出，静态条件下，总线节点的差动输入电压 $V_{diff.in}$ 由下述条件决定：

① 发送节点的差动输出电压 $V_{diff.out}$；

② 总线电缆的阻抗 $R_ω$，其中，$R_ω=\rho L$，ρ 表示电缆单位长度的特征阻抗；L 表示总线线路的长度；

③ 接收节点的差动输入阻抗（R_{diff}）。

在最差的情况下，发送节点在总线电缆的一端，而接收节点在总线的另一端。

为了计算接收节点在最差的情况下（即差动输入电压是最小）的差动输入电压，进行以下的假设和简化。

① 终端电阻 R_T 位于发送节点的输出处和接收节点的输入处；

② 所有其他节点都在传输线路的同一端，因此接收节点得到最小的差动输入电压。

计算电路框图如图 7-11：

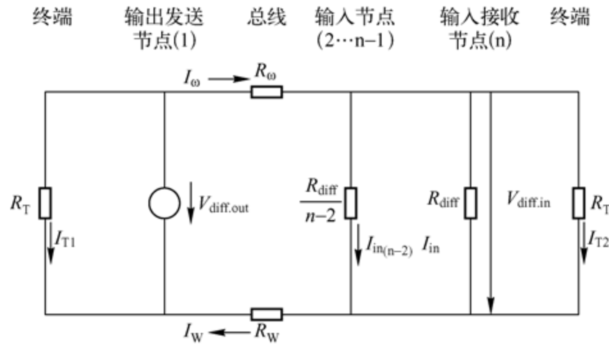


图 7-11　在接收节点计算差动输入电压的电路框图

接收节点有效的差动输入电压和发送节点的差动输出电压之间的关系，可以表示为

$$V_{diff.out} = V_{diff.in} + 2 \times R_{\omega} \times I_{\omega}$$

总线电流 I_{ω} 由下面的电流组成：$I_{in(n-2)}$（除了发送和接收节点的连接节点输入电流），I_{T2}（流过终端电阻的电流）和 I_{in}（接收节点的输入电流）。

由于 $I_{\omega}=I_{in(n-2)}+I_{T2}+I_{in}$，而且

$$V_{diff.in} = I_{in(n-2)} \quad \frac{R_{diff}}{n-2} = I_{T2} \quad R_T = I_{in} \quad R_{diff}$$

因此 $V_{diff.out}$ 和 $V_{diff.in}$ 之间的关系可用如下等式计算。

$$V_{diff.out} = V_{diff.in} + 2 \quad R_{\omega} \quad V_{diff.in}\left(\frac{1}{R_T} + \frac{n-1}{R_{diff}}\right)$$

于是可得到：

$$V_{diff.in} = \frac{V_{diff.out}}{1 + 2R_{\omega} \quad \left(\frac{1}{R_T} + \frac{n-1}{R_{diff}}\right)} \tag{7-4}$$

在 PCA82C250/PCA82C251 的通信过程中无需接收与发送的硬件转换控制，仅由软件来控制其转换过程，因而引入“显性”位和“隐性”位，它们的引入可在总线上实现非破坏性总线仲裁。如果差动输入电压低于 0.5V 或 0.4V，接收器将识别为隐性位，发送节点关断晶体管，这为具有“休眠”功能的系统提供了网络安全保障；如果差动输入电压高于 0.9V 或 1V 电平，接收器将识别为显性位，发送节点的输出晶体管导通，产生电流，相当于向总线传送或接收有效数据位。

要正确检测一个显性位，首先要求在接收节点有差动输入电压 $V_{diff.in.req}$，使得

$$V_{\text{diff.in.min}} \geqslant V_{\text{diff.in.req}}$$

差动输入电压 $V_{\text{diff.in.req}}$ 主要是由发送节点的驱动和网络的负载阻抗决定，可由接收器的显性阈值电压 V_{th} 和用户定义的安全余量给出。在最差的情况下，接收节点显性电平的最小差动输入电压一定要高于最差情况下输入晶体管的开关阀值加上一定的安全余量（系统要求），这个安全余量可认为是：检测显性电平的方程中发送节点的输出电平和接收器输入阈值之差的一个权值系数 k_{sm}。所以可以得到 $V_{\text{diff.in.req}}$ 的计算公式：

$$V_{\text{diff.in.req}} = V_{\text{th}} + k_{\text{sm}}\ (V_{\text{diff.out}} - V_{\text{th}})\text{，其中 } k_{\text{sm}} = 0 \sim 1 \tag{7-5}$$

由式（7-4）：

$$V_{\text{diff.in}} = \frac{V_{\text{diff.out}}}{1 + 2R_{\omega}\left(\dfrac{1}{R_{\text{T}}} + \dfrac{n-1}{R_{\text{diff}}}\right)}$$

为得到 $V_{\text{diff.in}}$ 的最小值 $V_{\text{diff.in.min}}$，使显性电平差动输出电压取最小值（$V_{\text{diff.out.min}}$），总线电缆阻抗取最大值（$R_{w.\text{max}}$），终端电阻取最小值（$R_{\text{T.min}}$），节点差动输入阻抗取最小值（$R_{\text{diff.min}}$），并且连接总线的最大节点数量为最大（n_{max}）。此时有：

$$V_{\text{diff.in.min}} = \frac{V_{\text{diff.out.min}}}{1 + 2R_{\omega.\text{max}}\left(\dfrac{1}{R_{\text{T.min}}} + \dfrac{n_{\text{max}}-1}{R_{\text{diff.min}}}\right)}$$

结合 $V_{\text{diff.in.min}} \geqslant V_{\text{diff.in.req}}$，可得

$$V_{\text{diff.in.min}} = \frac{V_{\text{diff.out.min}}}{1 + 2R_{\omega.\text{max}}\left(\dfrac{1}{R_{\text{T.min}}} + \dfrac{n_{\text{max}}-1}{R_{\text{diff.min}}}\right)} \geqslant V_{\text{diff.in.req}}$$

即：

$$\frac{V_{\text{diff.out.min}}}{1 + 2R_{\omega.\text{max}}\left(\dfrac{1}{R_{\text{T.min}}} + \dfrac{n_{\text{max}}-1}{R_{\text{diff.min}}}\right)} \geqslant V_{\text{th.max}} + k_{\text{sm}}\ (V_{\text{diff.out.min}} - V_{\text{th.max}}) \tag{7-6}$$

又因为 $R_{\omega} = \rho L \Rightarrow R_{\omega.\text{max}} = \rho_{\text{max}} L_{\text{max}}$；

上式可转换为

$$\frac{V_{\text{diff.out.min}}}{1 + 2\rho_{\text{max}} L_{\text{max}}\left(\dfrac{1}{R_{\text{T.min}}} + \dfrac{n_{\text{max}}-1}{R_{\text{diff.min}}}\right)} \geqslant V_{\text{th.max}} + k_{\text{sm}}\ (V_{\text{diff.out.min}} - V_{\text{th.max}}) \tag{7-7}$$

从而可得到关于总线电缆阻抗的最大总线长度 L_{max} 的公式：

$$L_{\text{max}} \leqslant \frac{1}{2\rho_{\text{max}}}\left(\frac{V_{\text{diff.out.min}}}{V_{\text{th.max}} + k_{\text{sm}} \times (V_{\text{diff.out.min}} - V_{\text{th.max}}} - 1\right) \times \frac{R_{\text{T.min}}\ R_{\text{diff.min}}}{R_{\text{diff.min}} + (n_{\text{max}} - 1)R_{\text{T.min}}} \tag{7-8}$$

以上分析了总线电缆电阻与 CAN 总线长度之间的关系，至于 CAN 网络中所能连接的最大节点数和因素有关，下面分析这个问题。

2．总线节点数量

对于 PCA82C250 和 PCA82C251，随着节点数量 n 的增加，负载阻抗变小，而驱动器输

出驱动能力有限，最小负载 $R_{L.min}$=45Ω，因此连接到网络上的节点数量由最小负载 $R_{L.min}$ 来决定。结合图 7-11 电路框图可计算节点的最大数量。

$$2(n_{\max}-1)\ R_{\omega.\max}+\frac{R_{T.\min}R_{diff.\min}}{(n_{\max}-1)R_{T.\min}+2R_{diff.\min}}>R_{L.\min} \qquad (7-9)$$

在接收端为最坏的情况（发送节点在总线电缆的一端，而接收节点在总线的另一端）下，可以认为总线电缆的阻抗 $R_{\omega}\approx 0$。此时，计算公式如下：

$$\frac{R_{T.\min}R_{diff.\min}}{(n_{\max}-1)R_{T.\min}+2R_{diff.\min}}>R_{L.\min}$$

从而，可得到最大节点数量 $n_{\max}$：

$$n_{\max}<1+R_{diff.\min}\left(\frac{1}{R_{L.\min}}-\frac{2}{R_{T.\min}}\right) \qquad (7-10)$$

式中，$R_{L.min}$ 表示驱动器 PCA82C250 和 PCA82C251 的最小输出负载，一般情况下 $R_{L.min}$=45Ω。

需要说明的是如果使用 PCA82C250，在驱动 R_L=45Ω的负载时，要求电源电压 V_{CC}>4.9V。利用上式可以得到不同差动输入阻抗、终端电阻、负载阻抗下的最大节点数量，如表 7-8 所示。

表 7-8　不同情况下的最大节点数量

最小差动输入阻抗 $R_{diff.min}$/kΩ	最小终端电阻 $R_{T.min}$/Ω	最小负载阻抗 $R_{L.min}$/Ω	最大节点数量
20	118	45	106
20	120	45	112

3．应用实例分析

以上分别分析了不同电缆阻抗对最大总线长度和最大节点数量的影响。下面就其应用进行具体分析。

（1）有关总线电缆横截面积的选择

例 3：由公式（7-8）和式（7-10）可分别计算得到不同电缆单位长度的特征阻抗对应的最大总线长度和节点数量。在假设 32 个节点 R_{ω}<21Ω；64 个节点 R_{ω}<18.5Ω；100 个节点 R_{ω}<16Ω情况下，表 7-9 给出了不同横截面积的总线干线电缆所对应的总线长度和节点数量。

表 7-9　干线电缆建议的最小总线横截面积

节点数量 / 总线长度/m	32	64	100
100m	0.25 或 AWG24	0.25 或 AWG24	0.25 或 AWG24
250m	0.34 或 AWG22	0.5 或 AWG20	0.5 或 AWG20
500m	0.75 或 AWG18	0.75 或 AWG18	1.0 或 AWG18

由上表可得出，在很多情况下，下接电缆可选择横截面积是 0.25～0.34mm²（或 AWG24、AWG22）的电缆。

（2）不同电缆对应的最大总线节点数量和总线长度

例 4：根据 ISO 11898 标准和 DeviceNet™规范，表 7-10 给出了电缆的特征阻抗，如下。

表 7-10　不同电缆的特征阻抗（1km=3280.84ft.，1ft.=0.3048m）

电缆类型	特征阻抗	
	ρ_{nom}/(Ω/km)	ρ_{max}/(Ω/km)
ISO 11898（汽车）；0.25mm²（或 AWG23）	70	90①
DeviceNet™细电缆	69	92
DeviceNet™粗电缆	18	23
0.5mm²（或 AWG20）	37	50①
0.75mm²（或 AWG18）	26	33①

①：假设值。

同时，利用已知典型数值：

最小显性电平值：$V_{diff.out.min}$=1.5V，最小差动输入阻抗：$R_{diff.min}$=20kΩ，要求的差动输入电压：$V_{th.max}$= 1.0V，最小的终端阻抗：$R_{T.min}$=118Ω，代入到式（7-8）：

$$L_{max} \leqslant \frac{1}{2\times\rho_{max}}\left(\frac{V_{diff.out.min}}{V_{th.max}+k_{sm}\ (V_{diff.out.min}-V_{th.max}}-1\right)\frac{R_{T.min}\ R_{diff.min}}{R_{diff.min}+(n_{max}-1)\times R_{T.min}}$$

可以计算得到不同总线电缆类型和连接在总线上节点数量不同的情况下，最大的总线长度。得出的结果见表 7-11。

表 7-11　不同电缆和不同总线节点数量 *n* 的最大总线电缆长度

电 缆 类 型	$L_{max}(k_{sm}$=0.2)/m			$L_{max}(k_{sm}$=0.1)/m		
	n=32	*n*=64	*n*=100	*n*=32	*n*=64	*n*=100
DeviceNet™（细电缆）/或 ISO 11898 电缆	200	170	150	230	200	170
DeviceNet™（细缆）	800	690	600	940	810	700
0.5mm²（或 AWG20）	360	310	270	420	360	320
0.75mm²（或 AWG18）	50	470	410	640	550	480

注意：如果要驱动多于 64 个节点或大于 250m 的总线长度，PCA82C251 电源电压 V_{CC} 的精度建议在 5%或更高。PCA82C250 在驱动 50Ω负载（即 64 个总线节点）时，需要至少 4.75V 的电源电压；驱动 45Ω负载（即 100 个总线节点）时，至少要 4.9V 电源电压。

7.3.3　总线终端和拓扑结构

CAN 总线的拓扑结构是指总线中输入输出节点的互连形式，根据连接形式的不同，CAN 网络中的拓扑结构主要有星型、环型、总线型及树型。其中最常用的结构是总线型拓扑和星型拓扑。具体拓扑结构分析如 1.2 节所述。

CAN 网络采用总线型拓扑有很多方面的优势，也得到了广泛的应用，CAN 高速标准 ISO 11898 通常使用总线结构作为网络的拓扑结构，有些情况下也会用到星型或其他类型的拓扑结构。现主要分析讨论总线终端处理方法。

根据 ISO 11898 协议规定，CAN 总线必须在网络的两端，通常是网络主控制器和网络最

远端的节点之间安装合适的总线终端电阻（通常是一个 120Ω的电阻）。其目的是为防止信号反射及提高线路 EMC 性能。

在 CAN 总线通信过程中，有两种原因导致信号反射：阻抗不连续和阻抗不匹配。

阻抗不连续是指信号在传输线传输过程中突然遇到电缆阻抗很小甚至没有，信号在这个节点就会发生反射。这种信号的反射与光从一种媒质进入另一种媒质的反射是类似的。消除这种反射的方法，是在电缆的阻抗突变处跨接一个与电缆的特征阻抗同样大小的电阻，使电缆的阻抗连续。由于信号在电缆上的传输是双向的，所以，在通信电缆的另一端可跨接一个同样大小的电阻。

引起信号反射的另外一个原因是数据收发器与传输电缆之间的阻抗不匹配。这种原因引起的反射，主要表现在通信线路处在空闲方式时，整个网络数据混乱。

为了提高网络节点的拓扑能力，CAN 总线两端需要接有 120Ω的抑制反射的终端电阻，它对匹配总线阻抗起着非常重要的作用，如果忽略此电阻，会使数字通信的抗干扰性和可靠性大大降低，甚至无法通信。在没有安装终端电阻或者终端电阻不合适的情况下，通信可能会出现以下非预期的行为：

1）低波特率情况下能够正确通信，但是在波特率较高时会出错。具体出错的波特率值取决于一系列因素，包括 CAN 网络长度、数据帧上携带的数据（它能够改变具体传输的最高频率以及网络附近的电磁干扰）等等。

2）引起 CAN 错误，包括 Form 错误、CRC 错误、Bit 错误、Stuff 错误等。

3）高频信号传输时，信号波长相对传输线较短，信号在传输线终端会形成反射波，干扰原信号。

在 CAN 总线 ISO 11898 规范中，已经将整个 CAN 网络的终端电阻简化为网络两端两个 120Ω的电阻。但是在实际使用过程中，由于总线结构的不同，并且有些节点具有一定的支线长度，需要对终端结构进行分析和改进，根据终端处理不同，分为分裂终端、多终端、单终端、终端不匹配四种情况，下面就具体总线终端进行分析，并给出了无端接电缆的下接长度。

1．分裂终端问题

分裂终端（split termination）可在不改变终端线路直流特性的前提下，增强 EMC 特性。其原理是，将每个终端电阻分成两个等值的电阻，即用两个 60Ω的电阻代替一个 120Ω的电阻，在两个等值电阻的中间抽头上得到共模信号。理想情况下，该共模信号即是直流信号，从而可通过电容将其接地。

一般情况下，电容取值 10～100nF，此时电容应连接到一个“静态”的地电平上。如果终端是放置在总线节点之内，建议将分离地连接到具有最低电感的模块连接器的地引脚上。

关于分离电阻的连接方法有两种，各有优缺点。

第一种情况：将两个终端电阻分别采用分离形式，并单独接地，如图 7-12 所示。这个方法可优化高频特性。但由于两个终端电阻都接地，可能会通过低电平形成干扰性的环流。

第二种情况：为避免第一种情况下的干扰性回路电流，可以只将一个终端电阻接地，如图 7-13 所示。这种接法在中频和低频范围的工作特性更好。

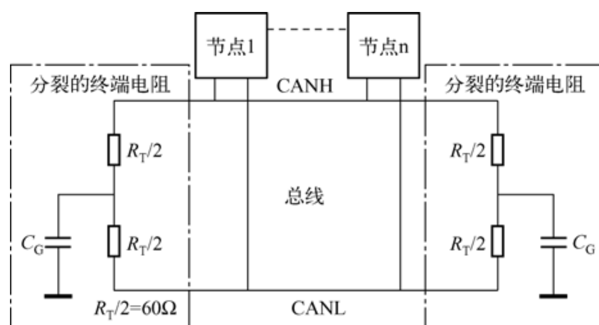


图 7-12　分裂终端的连接方法一

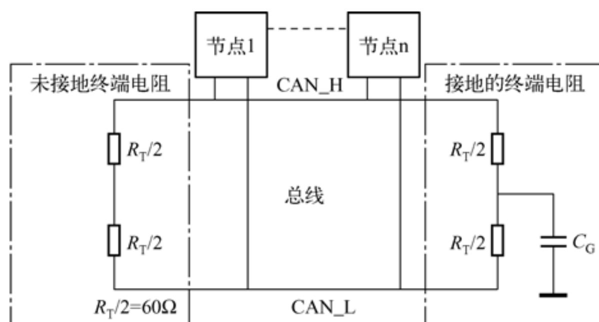


图 7-13　分裂终端的连接方法二

2．多终端问题

在一些应用中，需要使用和单线结构不同的拓扑结构，例如三分支的星形结构，如图 7-14 所示，采用的是三个等值电阻进行并联。为了使用这种拓扑结构，需要使用多终端的阻抗匹配方法。这个方法的原理是将总体终端阻抗即 60Ω，分配到两个以上的电阻上。对于三分支的星形拓扑结构，为使总体终端阻抗即 60Ω，则每个分支需要连接 3 倍总体终端阻抗值的电阻，即 180Ω。这个概念可以和分裂终端概念结合使用，因为它是和单线结构不同的网络拓扑结构。

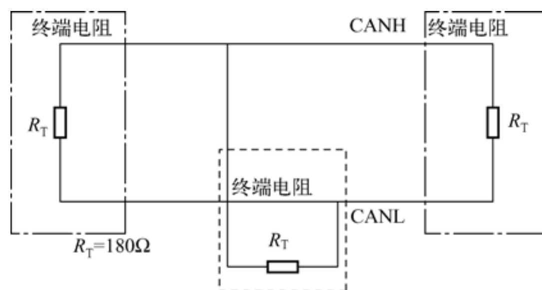


图 7-14　多终端概念

对于可选分支的拓扑结构，由于节点的可选择性可能会造成总体终端阻抗和驱动器输出驱动能力的不匹配。比如：对于可选节点的总线结构，如果干线的终端连接两个 180Ω的电阻，可选分支连接另一个 180Ω的终端电阻，于是，可选节点端接到总线时，特征阻抗与终端电阻

之间相互匹配。但是，当可选节点没有连接到总线时，特征电缆阻抗和终端电阻之间不匹配。

因此，对于可选分支的多终端问题主要处理方法是：

除了可选分支的终端外，其余分支的终端至少应保留终端数目的 50%。当所有的可选终端脱离总线以后，要求保留的基本终端电阻小于120Ω 。同时，整个总线的长度（包括所有的分支），应该小于给定配置的单线结构合适的总线长度。比如用三分支的星形结构代替100m 的单线结构，此时每个分支的终端要连接一个 180Ω的电阻而且每个分支的长度下降到小于 33m。

例 5：单线结构的 CAN 总线长度 $L = 100\text{m}$ ，终端电阻 $R_{\text{T}} = 120\Omega$ ，将其用三分支的星型结构来代替。则替代后的拓扑结构：

总线长度 $L = 33\text{m}$ ，终端电阻 $R_{\text{T}} = 180\Omega$ 。

3．单终端概念

如图 7-15 所示，CAN 总线网络中，仅有一个终端电阻位于主节点。从 CAN 位定时的角度考虑，只要留有足够的安全余量，即成功接收到报文的节点可在“应答间隙”期间发送一位显性位，以表示成功接收到一帧数据。则这个方法也是允许的。但为使 CAN 总线的数字通信的抗干扰性和可靠性不受到影响，凭经验，单终端接法的网络总线长度要小于正常终端接法总线长度的 50%。

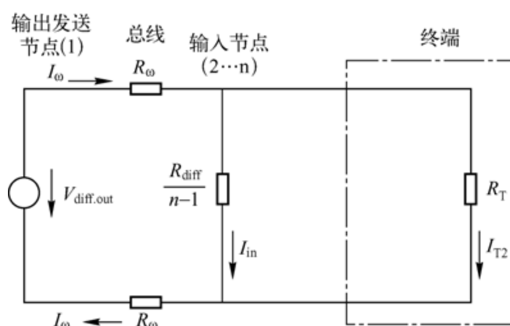


图 7-15　单终端概念

4．非匹配终端

这种方法有意使终端电阻与总线特征阻抗不匹配，当系统配置根据 CAN 位定时要求具有一个很大的安全余量时，即位速度或总线长度比标准终端概念的限制长度相对减少，可使用终端不匹配的方法。

该方法可减少对线路双绞线的要求，从而在同等配置下可增加总线驱动能力或降低功耗等。具体做法是将终端电阻 R_{T} 调高，大于电缆的特征阻抗，从而使总线相关的延迟显著上升。

采用这种方法与采用标准的终端接法相比，位速率或总线长度将会急剧下降。主要是由于当终端电阻增大时，由于两路总线线路传播延迟的值和总线时间常数有关，即和整个网络的电容量乘以有效的放电电阻的值有关，所以相应的总线延时将会急剧地增加，同时，这种不匹配方法的使用会使得总线节点之间地电平存在偏移，将会增加网络电容的放电时间。不过需要注意的是，差动终端电阻建议小于 500Ω，即所使用的位速率的上限阻抗是 2×1kΩ。

四种终端处理方法的比较：

表 7-12　终端处理方法的比较

终端类型	分裂终端	多终端	单终端	非匹配终端
方法原理	将每个终端电阻分成两个等值的电阻，在两个等值电阻的中间抽头上得到共模信号	将总体终端阻抗分配到两个以上的电阻上	仅在单端匹配终端电阻上	终端电阻 R_T 调高，大于电缆的特征阻抗，有意使终端电阻与总线特征阻抗不匹配
适用情况	总线型拓扑结构	总线型拓扑结构以外的结构	CAN 总线网络中，仅有一个终端电阻位于主节点	留有安全余量。并且对双绞线要求不高
优点	在不改变终端线路直流特性的前提下，增强 EMC 特性	实现多分支的终端匹配	留有足够的安全余量，这个方法也是允许的	减少对线路双绞线的要求，从而在同等配置下可增加总线驱动能力或降低功耗等
缺点	易形成干扰性的环流	对于可选分支，终端不容易准确匹配	总线长度小于正常终端接法总线长度的 50%	总线相关的延迟显著上升

5．无端接电缆的下接长度计算

CAN 总线的基本拓扑结构被近似看做总线结构，在很多实际应用中，总线节点并不是直接端接到总线电缆，而是通过没有端接的电缆连接到总线上，比如将诊断设备通过非终端的支线电缆临时连接到总线上时，由于无端接电缆的加入，相应的拓扑结构要进行一些改变。同时，由于无端接的下接电缆的存在，总线上将产生反射效应。但是由于网络提供了某种可靠性，如驱动器的滞后特性和 CAN 协议的重同步规则，发射不一定会产生干扰，从而反射波一旦到达总线终端就会消失，在一定的总线和支线长度下，是否能够容忍反射，主要取决于位定时参数。所以，反射是否被允许主要由位定时参数、干线电缆长度和下接电缆长度决定。

在位定时已知的情况下，下面主要计算下接长度和累积下接长度的上限。

对于下接长度，可凭经验利用下面的关系式来粗略计算无端接电缆剩余部分的长度 L_u：

$$L_u < \frac{t_{PROPSEG}}{50 \times t_p} \tag{7-11}$$

式中，$t_{PROPSEG}$ 是位周期传播段的长度，即时间段 1（ TSEG1）的长度减去重新同步跳转宽度（SJW）的长度；t_p 是每个单位长度特征电缆的延迟时间（如：5ns/m ）。

累积的下接长度是指所有支线电缆长度之和，对于累积的下接长度，可利用下述经验关系式粗略计算：

$$\sum_{i=1}^{n} L_{ui} < \frac{t_{PROPSEG}}{10 \times t_p} \tag{7-12}$$

由于无端接电缆的接入，总线的实际长度变长，因而为减小反射，总线线路实际的传播延迟应该是在考虑整个线路长度的基础上计算的。此时整个线路长度是干线电缆加上所有的下接电缆长度之和。所以，给定位速度下，下接电缆的接入，将有效地减少最大干线电缆长度。

下面举一个实例，分别计算无端接电缆的下接长度和累积下接长度：

例 6：已知位速度=500kbit/s，$t_{PROPSEG}$ =8×125ns=1000ns，t_p =5ns/m，则利用公式（7-11）和（7-12），可得：

$$L_{\mathrm{u}} < \frac{t_{\mathrm{PROPSEG}}}{50 \times t_{\mathrm{p}}} = \frac{1000\mathrm{ns}}{50 \times 5\mathrm{ns/m}} = 4\mathrm{m}, \quad \sum_{i=1}^{n} L_{\mathrm{ui}} < \frac{t_{\mathrm{PROPSEG}}}{10 \times t_{\mathrm{p}}} = \frac{1000\mathrm{ns}}{10 \times 5\mathrm{ns/m}} = 20\mathrm{m}$$

由此可得，在 CAN 传播段（PROP-SEG）长度是 1000ns 的情况下，一个非端接的下接电缆长度要满足 $L_{\mathrm{u}} < 4\mathrm{m}$，而累积的下接长度要满足 $\sum_{i=1}^{n} L_{\mathrm{ui}} < 20\mathrm{m}$。

7.4　CAN 总线的实时性问题分析

网络化控制系统（NCS）和传统控制系统相比，要求在串行实时总线上建立闭环控制回路，由此可见，NCS 对网络实时性的要求更高。为了满足系统的实时性要求，针对具体的应用情况，系统设计者一方面要选择合适的底层网络，另一方面，也要选择合适的高层应用协议。此外，还可以考虑网络结构的规划设计与优化，以保证较好地控制性能。对于每一个备选底层网络，要根据其协议规范和网络特点分析其实时性能，也就是分析其网络时延的来源、特性以及在不同网络环境下的表现等。本节以 CAN 总线为例，分析 CAN 总线的实时性能。

7.4.1　CAN 总线系统的实时性问题

CAN 采用基于"非破坏性位仲裁"的总线访问控制方式，虽然 CSMA/CA 方式可提高整个网络信息的传输效率，但该协议仅仅保证总线网络中最高优先级报文信息的实时传送，而低优先级的报文在发生总线访问冲突时均有一定的延时。一般来讲，不讲求实时性的网络是不适宜作为控制网络的，所以当需要构建如图 7-16 所示的典型网络控制系统时，CAN 能否满足控制系统对实时性能的苛刻要求，成为了首当其冲的问题。

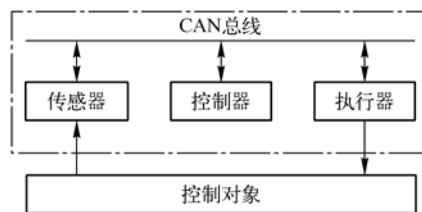


图 7-16　典型基于 CAN 总线的控制系统

本文将主要从网络延时和网络延时变化两个方面对 CAN 总线的实时性能进行分析。

7.4.2　CAN 总线延时分析

CAN 总线的网络延时包括：从待发送数据在总线节点 A 变化开始，直到其在另外一个节点 B 中得到确认，这期间的总时间延迟。根据 CAN 总线数据流的方向可以看出，CAN 的报文信息延时由帧延时、软件延时与 CAN 控制器延时、媒体访问延时等部分组成。

1）帧延时是信息串行化导致的延时，为最重要的延时因素。

2）软件延时是应用进程中，主 CPU 将数据从 CAN 控制器中读出/写入并作初步处理所耗费的时间，CAN 控制器的延时主要是 CAN 控制器为实现接收/发送缓冲器中的信息和串行化的信息的相互转化所开销的时间，另外还有收发器的延时。

3）媒体访问延时则是不同优先级报文抢夺总线资源时的总线冲突延时。

下面将对上述三种延时分别加以论述。

1．帧延时分析

帧延时即报文信息的传输延时，由帧长度和总线的传输速率决定。

根据 CAN 总线规范 2.0B，CAN 总线的报文信息共有以下四种帧类型：数据帧、远程帧、

错误帧和过载帧，其中最为重要的又是数据帧和远程帧，二者区别在于数据场的有无。而 CAN 的数据帧和远程帧又分为两种帧格式：标准帧和扩展帧，二者区别在于识别符的长短。所以帧类型和帧格式均会影响帧长度。

此外，CAN 总线为实现总线空闲的确定、CAN 控制器的同步与传输错误的检测，其控制协议规定，从帧起始到 CRC 序列结束（不包括 CRC 界定符），当出现连续的五个相同位信号，CAN 控制器将自动在其后添加一反相填充位，故填充位也可以影响帧长度。

所以帧长度由数据场、识别符以及填充位的个数联合决定。

CAN 总线中报文信息的传输速率是影响帧延时的另一重要因素。由于信息是串行发送的，故传输速率由波特率来度量。将 CAN 总线应用于工业现场中传感器和执行器的连接的时候，其连接的长度在 40～10000m 之间变化，相应传输速率在 1M～5kbit/s 之间变化。

当将 CAN 总线应用于工业控制现场时，应先综合控制网络的总体要求，计算最佳总线传输速率参数并进行预置，一般不容更改，从而构成帧延时中相对固定的因素。

综合上述帧长度、波特率和填充位的影响，针对扩展数据帧，得到其在最大传输速率条件下对应不同数据字节时的延时，如表 7-13 所示（波特率为 1Mbit/s，延时单位为微秒）。

表 7-13　扩展数据帧帧延时参数

数据字节长度/B		0	1	2	3	4	5	6	7	8
扩展帧的帧延时/μs	不含填充位	64	72	80	88	96	104	111	120	128
	含最大填充位	74	84	94	103	113	122	132	142	151

表 7-14 给出了标准帧和扩展帧之间的延时差。由于 CAN 总线的报文信息大部分是短帧信息，其传输的数据字节数较少，则识别符的差异导致的延时差异将达到 30%～40%，所以帧格式对延时信息的影响是巨大的。

表 7-14　标准帧和扩展帧延时差别

数据字节长度/B		0	1	2	3	4	5	6	7	8
延时差别（%）	不含填充位	45.5	38.5	33.3	29.4	26.3	23.8	21.7	20	18.5
	含最大填充位	46.2	38.7	33.3	29.6	26.4	24	21.8	20	18.5

2．软件及控制器延时分析

软件和控制器导致的延时与具体采用的主控 CPU、CAN 控制器和接口芯片有关。在本文所讨论的实际应用中，采用的主控器是 Intel 333MHz 的 CPU，CAN 控制器是 SJA1000，收发器是 PCA82C250。

为测量方便，采用的是一双 CAN 的 ISA 控制通信卡，一个 CAN 控制器作为发送节点，一个作为接收节点，排除了总线媒体访问的冲突延时。故只要测量总延时，并计算帧延时，就可进一步得到软件延时和控制器延时的值。

总延时包括从发送进程往 CAN 控制器的发送缓冲器中写第一个数据开始，一直到接收进程中将接收缓冲器中的有关数据全部读出的整个时间段。时间的测量可通过主控器控制主板上的 8254 计数芯片的计数通道 2 来获取，精度为 1μs，测量过程考虑填充位的影响，测量样本容量是 10000 次，延时参数取均值。

由表 7-15 可知，在固定发送速率条件下，随着发送字节数的递增，非帧延时也相应递增，这源于主控器与 CAN 控制器之间的数据交换量的增加，且 CAN 控制器实现信息串行化编码和解码的开销时间亦相应增加。

表 7-15　固定发送速率时对应不同发送字节数报文的延时

字节数		0	1	2	3	4	5	6	7	8
500kbit/s	总延时/μs	113.1	138.7	160.0	181.6	206.9	225.9	248.7	271.5	295.5
	非帧延时/μs	23.1	32.7.	40.0	45.6	52.9	55.9	64.7	71.5	79.5
50kbit/s	总延时/μs	944.2	1106.9	1264.2	1424.2	1607.4	1727.4	1902.2	2064.0	2247.3
	非帧延时/μs	44.2	46.9	44.2	44.2	67.4	27.4	62.2	64.0	87.3

从表 7-16 可以看到，对于发送同样的数据字节的报文，在发送位速率增加的情况下，从其非帧延时数据的相对稳定性可以看出，发送速率对非帧延时的影响有限。

表 7-16　固定的报文在不同发送波特率条件下的延时

位速率 bit/s		5k	20k	50k	100k	125k	250k	500k	1M
非帧延时/μs	1B 数据报文	200.3	55.3	46.9	37.7	34.4	33.5	32.7	32.1
	4B 数据报文	200.5	57.4	53.4	57.9	57.0	53.1	52.9	52.3
	8B 数据报文	201.0	105.2	87.3	87.0	80.3	80.2	79.5	79.2

综合上述二表可以得知，CAN 总线在高速通信时，发送数据字节数是影响非帧延时的主要因素。CAN 总线用于过程控制中时，因为对实时性要求比较高，则其通信速率一般≥50kbit/s，从而 CAN 网络的软件延时及控制器延时将在 30～100μs 之间变化。

3．媒体访问延时分析

据上面分析可知，如 CAN 总线中的网络节点数量很少且网络负载很小，那么网络信息延时基本上是由帧长度、位速率、应用进程和控制器决定。

但是随着工业控制系统自动化程度的提高，其复杂性也随之提高，这样系统中的控制节点将越来越多，而控制网络中信息流也将急剧增加。在这样的一个多节点、高负载的网络控制系统中，由报文抢占总线资源而引起的媒体访问延时将凸显其重要性。因此对媒体访问延时的分析对于设计工业控制系统来说，将具有非常重要的现实意义。

对于一个总线型控制网络，要对其媒体访问延时性能进行分析，需建立一个合理的数学模型进行理论分析，并进行离散时间系统的仿真试验加以验证。

（1）网络建模与分析

根据 CAN 规范 2.0B，CAN 总线可视作一个带优先级的排队系统。假设其系统容量无限、顾客来源无限（当系统容量较大时，引入的误差很小），服务策略采用不占先服务(Non-preemptive)。

设 CAN 总线中的报文依优先级分为从 0 到 v-1 的 v 个等级。对应识别符与优先级的关系，设定等级 i 越小的报文优先级越高。各个等级的报文均以泊松模式进入系统，其平均到达速率分别为 $\lambda_0,\lambda_1,\cdots,\lambda_{v-1}$。系统对各个报文的平均服务速率分别为 $\mu_0,\mu_1,\cdots,\mu_{v-1}$，服务时间服从指数分布，具备马尔可夫特性。

则利用排队论的分析方法，可得到第 i 级报文的等待时间为

$$T_{\omega_i} = \frac{\sum_{j=0}^{v-1} \lambda_j / \mu_j^2}{(1-\sigma_{i-1})(1-\sigma_i)}$$

$$\sigma_{i-1} = \sum_{j=0}^{i-2} \frac{\lambda_j}{\mu_j}, \quad i=0,1 \ldots, v-1$$

利用 Little 定理可以得到平均等待时间，平均等待队列长度等其他性能参数，这里不加赘述。本书有关网络媒体访问延时的性能参数均用平均等待延时度量。

这里选定含 32 个网络节点的控制网络作为建模对象，其报文优先级分布在 0 到 31 之间；各个优先级的报文的到达速率相等；相应服务速率均为单位时间。图 7-17 为不同负载条件下对应各优先级报文的等待延时曲线。

（2）网络冲突仿真与分析

在本仿真中，CAN 总线由事件驱动，是一典型离散事件系统。网络报文优先级仍设定在 0～31 之间，针对 CAN 总线的特点，仿真策略采用事件调度法。仿真的采样样本容量为 10 万个报文信息，延时单位为单位位延时。

结合图 7-17、图 7-18 可知，经网络冲突延时的仿真所获取数据和前述建模分析结论吻合，基本验证了模型的正确性，说明所建立网络模型的延时分析对网络系统的设计具有现实的指导意义。

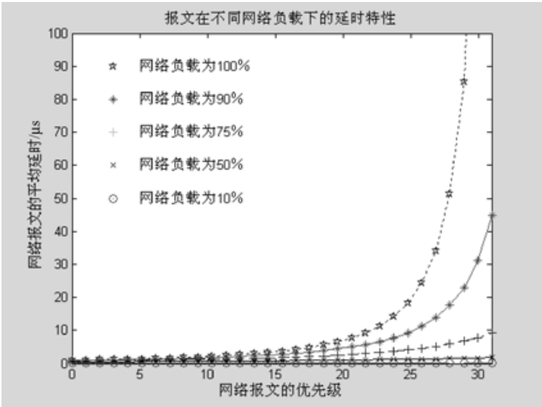


图 7-17　网络模型的延时性能

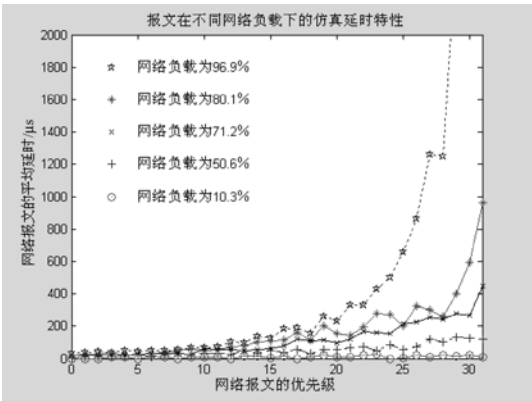


图 7-18　仿真得到的延时性能

从图 7-18 可以看出，网络负载对报文信息的等待延时有着巨大的影响；当网络负载较小（<50%）的时候，各优先级报文的延时没有明显差别，特别是当网络负载<10%时，CAN 总线几乎能保证各个优先级报文的实时发送。

表 7-17　仿真实验得到的网络性能参数

网络性能参数列表				
网 络 负 载	队列最大长度	队列平均长度	最大等待时间/μs	平均等待时间/μs
10.311 %	2	0.009	316	9.59
36.267 %	3	0.096	707	29.68

（续）

网络性能参数列表				
网 络 负 载	队列最大长度	队列平均长度	最大等待时间/μs	平均等待时间/μs
50.636 %	4	0.231	985	49.10
71.166 %	8	0.807	5289	112.00
96.866 %	27	9.113	43474	778.33
99.987 %	912	467.729	2705244	24775

同时优先级对等待延时影响亦是巨大的，从图 7-15 可看到，当网络负载较高且各优先级的报文分布比较均衡的时候，优先级 $i<v/2$ 的报文基本能得到及时发送，但随着优先级继续增加，其网络延时剧增，将无法满足控制网络对实时性的要求。

总的来说，对于一个真正的工业控制网络系统，网络不可能完全是由报文事件驱动的，其有很大一部分是由时间驱动的周期性数据报文，故而上面所作的分析可以视为所设计的控制系统在最恶劣的网络条件下网络延时的一个分析方法。

7.4.3 CAN 总线延时变化分析

延时的变化有诸多方面的原因：首先是由于媒体访问冲突导致的延时变化；其次是由于填充位的变化导致的延时变化；另外还有诸如器件引起的延时变化等。

1．媒体访问冲突导致的延时变化

上面已经分析，网络平均负载对平均等待延时影响巨大。但是即使在同一平均网络负载条件下，由于微观网络负载的差异，将导致完全相同的两个报文的延时有巨大差异。一个极端的情况是某一时间段网络微观负载为零，报文信息立即得到发送；但是在另外一个时间段，网络的微观负载为满负荷运行，如果该报文优先级不占优势的话，这将导致极大的延时。

图 7-19 为网络冲突仿真图，各优先级报文的延时变化的大小用同优先级报文的最大最小延时差值来度量。由图 7-19 可知，网络的平均负载对延时变化的影响是巨大的，随着平均负载的提高，报文的最大最小等待延时的差值随之增加。但其中高优先级的报文的延时变化对网络负载不敏感。

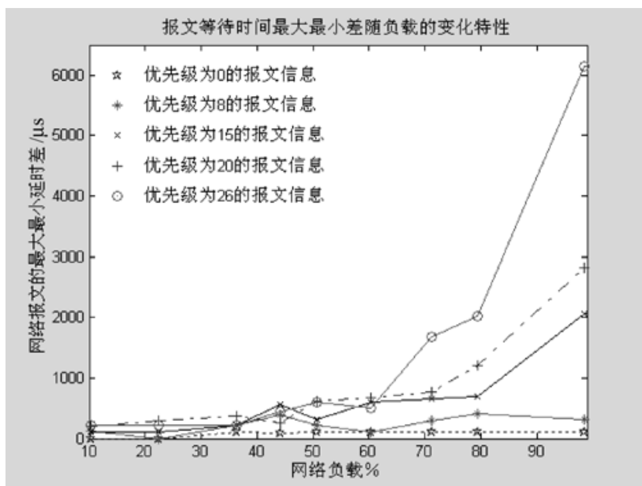


图 7-19　不同网络负载下的报文延时变化

2．填充位导致的网络延时变化

在实际应用中间，填充位可导致 0%～15%的帧延时波动。由于其影响是随机的，所以只能从统计特性上对填充位的影响加以分析。

在实际仿真中，针对使用几率最高的标准数据帧进行仿真试验，设定采样样本容量为 100 万；同时在仿真过程中，应注意报文中控制信息、数据信息与 CRC 场的耦合关系，以及一些特殊的位约束。

从表 7-18 填充位的分布数据列表可以看到，填充位主要集中在 1～5 个之内，具有较好的集中特性。因实际应用当中，报文的优先级和发送字节相对固定，报文的数据场变化限定在某些具体范围内，这样也使得填充位的变化相对较小。

表 7-18　典型数据帧的填充位分布

填充位个数		0	1	2	3	4	5	6	7	8	9	10
填充位	1B 数据	0.0	30.2	41.4	21.8	5.75	0.76	0.11	0.01	0.00	0.0	0.0
	4B 数据	5.78	20.7	30.6	25.0	12.5	4.27	0.93	0.14	0.01	0.00	0.00
	8B 数据	2.0	10.1	21.1	25.5	20.7	12.5	5.58	1.87	0.51	0.10	0.01

从表 7-19 的统计特性上看，对于发送不同数据字节的数据帧来说，其等效填充位的变化相对平缓。

表 7-19　不同报文的平均填充位数

网络数据长度/B	0	1	2	3	4	5	6	7	8
平均填充位数	2.30	2.05	2.28	2.54	2.35	2.54	2.80	3.25	3.25

综合上面两表可知，由填充位导致的延时变化处于延时因素中的从属地位。

3．其他延时变化

在实际测量应用中发现，即使在完全相同的测试条件下，同一对节点之间报文发送方向的差异也将导致较为明显的延时变化。该延时变化主要由控制器以及外接晶体振荡差异所致。另外还发现即使是发送方向也相同，所得到的报文延时仍呈一定的散布。此两种延时变化原理上不可避免，但其不受网络负载的影响，且影响较小。

7.4.4　实时性能提升策略

对于一个典型的控制网络来说，其信息响应时间要求为 0.01～0.05s。而通过上述对 CAN 总线实时性能分析可知，网络带宽和网络负载对控制网络实时性能有着巨大的影响。当一个 CAN 总线的传输速率≥100kbit/s，其网络负载≤50%，且网络运行状况平稳，则完全能够保证控制信息在 1～10ms 内得到准确传送。如果进一步提高网络传输速率，使其≥500kbit/s，降低网络负载，使其≤10%，则控制报文甚至可在 1ms 之内得到实时传送。这样，对于一般的过程控制，CAN 总线网络完全能够满足其实时性要求，从而使得在其基础之上构建一个具备高实时性的控制网络系统成为了可能。当然这在实现器件、网络负载、网络传输的平稳性能以及控制网络系统容量方面均提出了自己的要求。

通过如上对 CAN 实时性能的分析，可为设计控制网络系统提供了如下参考：

1）当标准帧能满足系统对控制容量、传输可靠性等性能需求时，尽量避免使用扩展帧。

2）在满足控制系统稳定性的前提下，尽量提高控制网络的传输速率，增加带宽。

3）尽量减小控制网络中不必要的节点及报文信息，降低网络负载，以预留较大的网络带宽余量。

4）尽量选取性能稳定、均一的器件构建网络硬件，以提高网络的整体性能。

5）可适当增大控制采样周期，尽可能采用同步传输方式，并避免网络的微观拥塞情况。

总之，CAN 总线网络作为一种优良的现场总线控制网络，凭借其在实时性和可靠性等方面的优异性能，将在工业现场控制等领域得到更为广泛的应用。

7.5 本章小结

本章主要针对 CAN 总线在应用中的一些问题展开介绍，给出了在扩展模式下，SJA1000 的功能分析，详细介绍了 CAN 总线滤波器的配置和 CAN 总线的驱动问题，针对 CAN 总线应用中的实时性问题，给出了 CAN 总线的提高实时性能的策略。

思考题与习题

7.1 简述 CAN 总线的扩展模式的主要功能及其作用。

7.2 简述在 PeliCAN 模式下验收滤波的配置。

7.3 分析 CAN 总线的线路长度及节点数计算方法。

7.4 CAN 总线的网络延时包括哪些部分？

7.5 简单分析帧格式对延时信息的影响，并分析影响非帧延时的主要因素。

第 8 章　CAN 总线的应用层协议

以 CAN 总线的物理层和数据链路层为基础，根据应用对象的特点可以定义不同的 CAN 应用层协议，形成物理层、数据链路层和应用层三层为主的现场总线结构。应用层协议为系统组网、设备开发和通信提供了更高层次的执行标准和规范。针对特定系统制定的应用层协议一般不具备普适性，但随着 CAN 总线技术的广泛应用，一些应用层协议逐步发展成为行业标准并推广开来，其中以 CANopen 协议和 DeviceNet 协议为代表。应用层协议的标准化和推广使基于 CAN 总线的现场总线的组网和综合应用变得非常简便。

本章要点

本章主要介绍 CAN 总线的应用层协议，主要内容如下：

- CANopen 协议；
- DeviceNet 协议；
- CAN 总线其他应用层协议简介；
- CAN 总线应用层协议对比分析。

8.1　CAN 总线的应用层协议概述

CAN 总线的应用层协议一般是某个公司或组织根据特定对象定义并使用的，比如 CANopen 协议最初是德国 BOSCH 公司为解决产品部件内部网络通信而开发的。随着协议的完善和推广，一些有代表性的应用层协议逐步形成行业标准并在更为广泛的领域内使用，其中以 CANopen 协议和 DeviceNet 协议为代表。

CANopen 协议主要适用于嵌入式网路，具有非常大的自由度；DeviceNet 协议主要适用于工业设备内部组网，是一种低成本的通信总线。CANopen 协议成功应用于机床控制、军事、航海、医疗、铁路等多个领域，而 DeviceNet 协议已成为工业控制领域的领导者之一。

8.2　CANopen 技术协议

CANopen 协议最初的目的是研发一种标准化的嵌入式网络，方便不同设备接入到系统中，使系统具有非常灵活的自由度。经过 CiA（国际用户与制造商团体）的完善和推广，CANopen 协议获得了广泛的认可并大量应用于嵌入式技术领域。

CANopen 协议的主要技术参数：

① 网络规模：支持 127 个节点；

② 网络长度：25～5000m（与波特率有关）；

③ 通信速度：10kbit/s～1Mbit/s；

④ 总线型拓扑：中继、级联；

⑤ 寻址方式：主/从、对等、广播和多主；

⑥ 系统特性：节点移除、面向请求/响应的网络通信功能、大量信息的数据组帧功能。

为实现嵌入式网络的设计目标，CANopen 协议定义了“对象字典”实现设备功能的标准化描述，并使用“文档”对设备和设备配置进行标准化描述。为使设备可用于任何 CANopen 网络，从以下三个层次保证网络的兼容性要求：

① 应用层：新设备入网后不能影响网内其他设备的通信，即设备的一致性要求；

② 通信协议层：设备可与网内的其他设备通信，即设备的互用性要求；

③ 设备子协议层：同类设备之间可相互替换，即设备的互换性要求。

8.2.1 对象字典

CANopen 协议通过对象字典（Object Dictionary，OD）实现设备的标准化描述。对象字典是 CANopen 中的一个核心概念，在其他一些现场总线（如 Profibus、Interbus-S 等）系统中也使用了类似的设备描述形式。

对象字典包含了所有通过网络访问的参数，例如：设备标识符、生产商名、PDOs 和 SDOs 的通信参数、设备监控（“error control”）等都保存在对象字典的通用区。对象字典的设备描述区则包含了 IO 功能（开关量和模拟量的输入和输出）、设备参数、PLC 映射等内容。如果对象发生错误，对象字典还可以配置其行为。

CANopen 协议将对象字典进行了分配，表 8-1 给出了对象字典的通用结构。

表 8-1　CANopen 对象字典通用结构

索　引	对　象
0000	保留
0000～001F	静态数据类型
0020～003F	复杂数据类型
0040～005F	制造商定义的复杂数据类型
0060～007F	设备子协议定义的静态数据类型
0080～009F	设备子协议定义的复杂数据类型
00A0～0FFF	保留
1000～1FFF	通信对象
2000～5FFF	制造商定义的对象
6000～9FFF	标准的设备子协议区
A000～AFFF	符合 IEC61131-3 的网络变量
B000～BFFF	用于路由器/网关的变量
C000～FFFF	保留

使用对象字典可以描述对象的各类属性。其中静态数据类型描述数据类型的定义，如整

型、浮点型等；复杂数据类型描述了 4 种预定义的数据结构；制造商定义的复杂数据类型是制造商特定的复杂数据结构；设备子协议定义的静态数据类型和复杂数据类型描述的是设备子协议中的数据类型；通信对象描述了设备通信的特征；制造商定义的对象描述了制造商自定义的且在设备子协议中没有规定的特定设备对象特征。

8.2.2　CANopen 子协议

CANopen 协议由通信子协议（Communication Profile）和设备子协议（Device Profile）等子协议文档组成。通信子协议描述了对象字典的形式和通信子协议区中的对象及通信参数，同时描述了 CANopen 通信对象，如 CiA-DS301 通信子协议。如前所述的对象字典，设备子协议定义了设备在对象字典中的对象，同时描述了具体对象的索引、子索引、名字、功能和数据类型，以及对象的只读/只写/读写、必选/可选等属性，如 DS401 和 DS402 等设备子协议。

CANopen 使用对象字典对设备功能实现了标准化的描述，通过字典的入口可以完成对设备的网络访问。在 CANopen 网络中的所有节点都有一个与之对应的对象字典，节点自身可选择完全按照对象字典描述或者只提供其中 CANopen 协议规定必须的对象。

表 8-2 和表 8-3 给出了常见的一些通信子协议和设备子协议的功能。

表 8-2　CANopen 通信子协议

通信子协议	功　能
CiA-DS301	定义 CANopen 应用层规范和通信规范，包括对象字典和服务数据、过程数据、网络管理对象等
CiA-DS302	定义网络启动步骤、主节点和管理节点的定义、可编程设备的输入/输出定义、冗余通信方式等
CiA-DS304	CANopen 安全相关的通信框架

表 8-3　CANopen 设备子协议

设备子协议	功　能
CiA-DS401	本地 I/O 模块的设备描述，定义了通用 I/O 模块的设备规范，主要定义了对象字典中 6000H 到 6FFFH 之间的内容
CiA-DS402	定义了运动控制的设备规范
CiA-DS404	CANopen 检测设备和闭环控制器的设备描述
CiA-DS405	符合 IEC61131-3 便准的可编程设备的设备和接口描述，定义了 IEC61131 标准设备规范
CiA-DS406	CANopen 编码器的设备描述

8.2.3　CANopen 数据传输机制

CANopen 使用了两种基本数据传输机制：进程数据对象（Process Data Object，PDO）和服务数据对象（Service Data Object，SDO）。PDO 实现小数据块的高速交互，一个 PDO 最长可包含 8B 的数据，一般采用事件触发、循环或请求方式发送；SDO 主要完成设备配置过程中的参数传输和大数据块的交互。

CANopen 协议定义了 4 种报文（通信对象）：PDO、SDO、预定义和网络管理报文。

（1）PDO

用于数据的实时交互，主要使用事件驱动、远程请求或轮询、同步传输等触发方式启动PDO 传输及接收。

（2）SDO

用于读写设备的对象字典，可传输任意长度的数据，但需要额外的协议开销。

（3）预定义或者特殊功能对象

主要包括同步（SYNC）、时间标记对象（Time Stamp）、紧急事件（Emergency）、节点/寿命保护（Node/Life guarding）、Boot-UP 等。

（4）网络管理

符合 CAL 中的 LMT、NMT 和 DBT 服务部分，实现初始化、设备配置、保护和管理等功能。

CANopen 具有安全传输功能，使用安全相关服务（Safety-Relevant Services）可通过安全相关数据对象 SRDO（Safety Relevant Data Object）来实现安全通信。

8.3 DeviceNet 协议

8.3.1 DeviceNet 概述

DeviceNet 协议是 Allen-Bradley 公司为解决设备内部部件组网问题而开发制定的应用层协议，后由 ODVA（Open DeviceNet Vendor Association）组织进一步完善，并在市场推广使用。DeviceNet 协议主要适用于工业自动化控制领域，随着 CAN 总线技术在工业领域的广泛使用，目前 DeviceNet 协议已成为工业控制领域的领导者之一。

DeviceNet 是一种低成本的通信网络，可以将工业设备的不同部件，如传感器、执行器、人机交互接口等，通过 CAN 总线连接到一起，改变了原来依靠硬连线的部件连接方式。使用 DeviceNet 组网，可以使不同厂商提供的同类部件具有良好的互换性，解决了部件接口差异带来的安装和调试问题。DeviceNet 协议是开放的，而且免费提供给设备供应商使用，可以从 ODVA 组织获得 DeviceNet 协议并获得技术支持。

DeviceNet 的主要特点：

① 部件级低端设备的低成本网络连接方案；

② 可使部件级设备具备一定的智能化水平；

③ 支持主/从通信和对等通信方式。

DeviceNet 的主要物理特性：

① 采用主干线和分支线联合的网络结构；

② 最多支持 64 个节点；

③ 可在线解除节点；

④ 同时支持网络供电及自供电设备；

⑤ 使用密封或开放式连接器；

⑥ 支持接线错误保护；

⑦ 可选的数据速率为 125kbit/s、250kbit/s 和 500kbit/s，见表 8-4；

表 8-4　通信速率对总干线、支线长度的影响

数据速率/（kbit/s）	干线长度/m	分支线长度	
		最大值/m	累积（不超过）/m
125	500	6	156
250	250		78
500	100		39

⑧ 可调整电源结构以满足分类应用的需求；
⑨ 每个电源的最大容量可以达到 16A；
⑩ 在带电状况下可以完成操作；
⑪ 电源接口可以连接符合 DeviceNet 标准的供电装置；
⑫ 内置过载保护；
⑬ 供电主干线包括了电源线和信号线（如图 8-1 所示）。

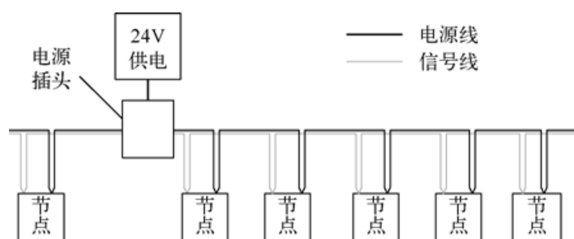


图 8-1　DeviceNet 总线主干线示意图

8.3.2　DeviceNet 对象模型

使用对象模型为模板，DeviceNet 可完成对部件组件的属性、服务以及行为的管理和实现。

（1）对象编址

对象模型使用 4 个 ID 编址的方式为每个属性提供寻址方案，分别为节点标识符（MAC ID）、类标识符（Class ID）、实例标识符（实例 ID）和属性标识符（Attribute ID）。

节点标识符（MAC ID）：一个整数标示值，分配给每个节点，用于区分同一连接上的不同节点，如图 8-2 所示。

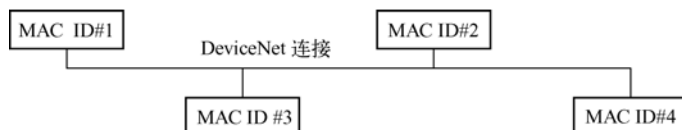


图 8-2　介质访问控制标识符

类标识符（Class ID）：一个整数标识值，分配给网络上可访问的每个对象类。

实例标识符（实例 ID）：一个整数标识值，分配给每个对象实例，用于识别相同类中的不同实例。

属性标识符（Attribute ID）：赋于分类或实例属性的整数标识值，如图 8-3 所示。

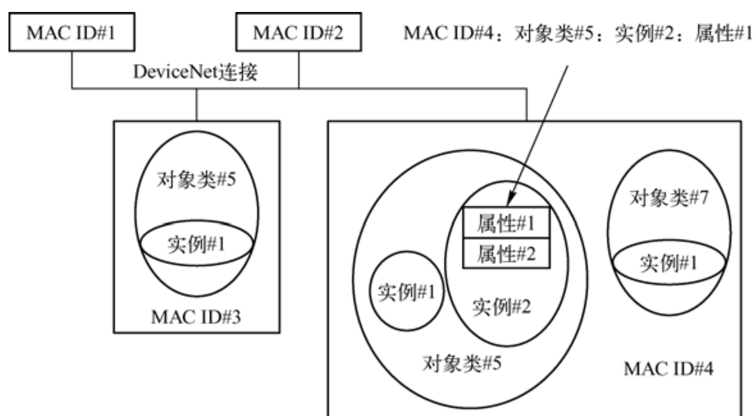


图 8-3　属性标识符示意图

服务代码（ServiceCode）：表示一个特定的对象实例或对象分类功能的整数标识值。具体如图 8-4 所示。

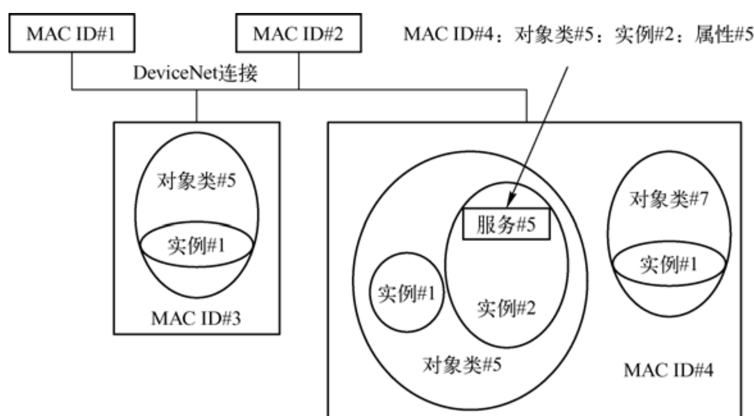


图 8-4　服务代码示意图

（2）寻址范围

开放部分（Open）：对于所有 DeviceNet 使用者来说都是通用的，取值范围由 ODVA 定义。

制造商专用（Vendor Specific）：在开放部分定义的有效范围之外，制造商可扩展其设备的功能，取值范围由设备制造商定义。

对象类专用（Object Class Specific）：用于服务代码的定义，取值范围按 Class ID 定义。

表 8-5 定义了 Class ID 有效的取值范围。

表 8-5　Class ID 的有效取值范围

范　围	含　义
00～63hex	开放部分
64～C7hex	制造商专用
C8～FFhex	DeviceNet 保留、备用
100～2FFhex	开放部分

（续）

范　　围	含　　义
300～4FFhex	制造商专用
500～FFFFhex	DeviceNet 保留、备用

表 8-6 定义服务代码有效的取值范围。

表 8-6　服务代码有效的取值范围

范　　围	含　　义
00～31hex	开放部分，为 DeviceNet 的公共部分
32～4Ahex	制造商专用
4B～63hex	对象类专用
64～7Fhex	DeviceNet 保留、备用
80～FFhex	非法/不用

表 8-7 定义属性 ID 有效的取值范围。

表 8-7　属性 ID 有效的取值范围

范　　围	含　　义
00～63hex	开放部分
64～C7hex	制造商专用
C8～FFhex	DeviceNet 保留、备用

表 8-8 定义属性 ID 有效的取值区间。

表 8-8　属性 ID 有效的取值区间

区　　间	含　　义
00～63（十进制）	MAC ID。如果没有分配其他值，那么设备初始化时的默认值为 63（十进制）

8.3.3　DeviceNet 的连接方案

为实现所有应用的通信， DeviceNet 连接提供了一个在多端点之间的通信路径，端点为应用程序。连接建立后，与连接关联的传输赋予一个连接标示值（CID）。DeviceNet 的基于连接的方案可以建立以下的两种类型的连接：

（1）I/O 连接（I/O Connections）

在生产或消费应用之间提供了专属的、具有特定用途的通信路径。特定的 I/O 数据通过这些路径传输。I/O 报文中包含一个连接 ID 及 I/O 数据，I/O 报文内数据的含义隐含在相关的连接 ID 中，而连接的端点可识别 I/O 报文的用途或含义。

（2）显式报文连接（Explicit Messaging Connections）

显式报文的连接在两个设备之间提供了一般的多用途的通信路径，显式报文是通过显式报文连接进行交换的。显式报文的含义和用途是在 CAN 的数据块中确定的。显式报文提供了

执行面向典型请求/响应引导功能的方法（例如模块配置），一个显式报文包含一个连接 ID 和相关联的报文协议。

8.3.4 DeviceNet 报文协议

1．显式报文

显式报文使用 CAN 帧的数据区来传递 DeviceNet 定义的报文。如图 8-5 所示：

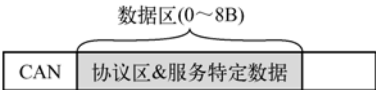


图 8-5　显式报文 CAN 数据区的使用

图 8-6 为显式报文数据区的格式。完整显式报文数据区包括：报文头和完整报文体。如果报文的长度大于 8B，则在 DeviceNet 上分段传输。连接对象需提供分段和重组功能。一个显式报文的分段包括：报文头、分段协议和分段报文体。

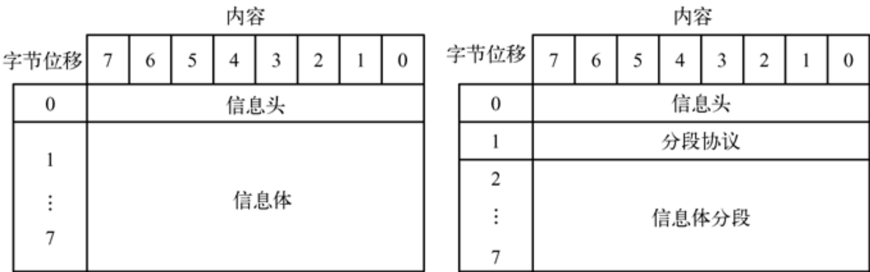


图 8-6　显式报文数据区格式

2．输入/输出报文（I/O 报文）

除了能够被用于发送一个长度大于 8B 的 I/O 报文的分段协议，DeviceNet 不在一个 I/O 报文的数据区内定义任何有关报文的协议，如图 8-7 所示 。

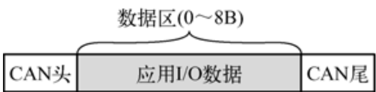


图 8-7　一个 I/O 报文的数据区

3．分段/重组

由于 CAN 总线帧的最大数据长度为 8B，一个长度大于 8B 的报文必须使用分段及重组的方法进行传送。在 Device 中分段/重组功能由连接对象提供，且支持分段发送及接收是可选的。对于显式报文连接和 I/O 连接而言，触发分段发送的条件是不同的。

显式报文检查需要发送的报文的长度。如果报文长度大于 8B，那么就使用分段方式传输。I/O 连接是检查连接对象的 produced_connection_size 的属性，如果大于 8B，那么使用分段方式传输。

4．重复 MAC ID 检测协议

DeviceNet 中节点标示符（MAC ID）用于区分不同节点，每一个节点需分配一个 MAC ID。由于配置过程由人工设置，在同一链接上可能会出现不同模块使用相同 MAC ID 的情况。所以要求 DeviceNet 模块参与重复 MAC ID 的检测算法。在组 2 中定义了一个特定的报文 ID 值来指定重复 MAC ID 检查报文，如图 8-8 所示。

标志位											信息 ID 含义
10	9	8	7	6	5	4	3	2	1	0	
1	0	MAC ID						组 2 信息			组 2 信息
1	0	目的 MAC ID						1	1	1	重复 MAC ID 检查信息

图 8-8　重复 MAC ID 检查 CAN 标识区

5．显式报文客户机应用考虑

使用通过显式报文连接时，请求/响应超时或者分段转发超时都可能会影响发送显式请求报文的能力。在客户机应用程序设计中可考虑增加对响应超时的处理，同时处理最后分段应答超时问题。

DeviceNet 协议最基本的功能是实现设备内部部件级节点的数据交互。因此，这种通信是基于面向连接的（点对点或多点传送）通信模型建立的。这样，DeviceNet 既可以工作在主从模式，也可以工作在对等模式。

8.4　CAN 总线其他高层协议

8.4.1　CANaerospace 协议

CANaerospace 协议除了特定的通信协议外，还提出了很多涉及航空电子安全的标准。航空航天使用的系统处理方式及设计标准对 CANaerospace 协议的制定产生了较大的影响。因此，CANaerospace 为保证实现最佳的电磁兼容性，除协议本身外，还制定了包括像连接器，数据线，以及设计向导等硬件处理的规范。

在航空航天系统中使用网络技术要求论证网络在所有条件下数据传输的正确性，这就要求 CANaerospace 协议保持简易性设计。在 CANaerospace 协议提出之前，几个基于 CAN 总线的工业应用层协议被用于航空应用的可用性研究，然而这些协议的设计无法满足航天航空的具体要求，如系统冗余，单模故障保护以及系统启动时间的限制等。此外，为了适应不同工业应用的需求，基于 CAN 的工业应用层协议非常复杂，不适于航空航天的简易性设计要求。

CANaerospace 协议是一种开放的，免费使用的应用层协议。Euro copter 直升机制造商采用该协议并作为系统通信总线标准应用于在全天候救援直升机中。CANaerospace 的 1.6 版本被美国国家航空和宇航局（NASA）在 2001 年作为“AGATE 数据总线”的标准。清华紫光基于 CANaerospace 开发了 Aero Vodochody/Ibis Ae270 飞机综合航空电子设备系统（SAM）。

8.4.2　CANKingdom 协议

CANKingdom 是一种 CAN 应用层协议，或者说是一种中介协议更为准确。在 CAN Kingdom 系统中，任何单一节点中都不需要网络知识，系统设计者对系统负全责。例如：为了组建一个通用的即插即用系统，系统设计者需要详细说明在哪种环境中配置哪些节点。

CANKingdom 满足了开发人员设计通用模块的需求，并不需要提前确定模块将集成到哪个系统中，以及它所适应的 CAN 高层协议的类型。

CAN 报文识别符不但识别报文而且还起到支配总线访问的作用，且发送和接收模块中数据域的数据结构是相同的。CANKingdom 协议通过采取一些简单的设计规则优化系统通信，包括容许有些不遵循 CANKingdom 协议的模块接入到 CANKingdom 系统中。

8.4.3 SDS 协议

SDS（Smart Distributed System）协议是由 Honeywell 公司开发，用于工厂自动化控制系统的 CAN 应用层协议。应用 SDS 可将传感器、执行器、控制器及其他工业设备连接为一个整体，该协议适用于高速、实时的分布式控制系统中。

SDS 有两种报文传输格式：短格式和长格式。短格式报文不包括任何数据字节，用于高效率的接口，是一个与单个二进制 I/O 设备进行通信的接口。长格式支持不分段或分段传输。

8.5 CAN 总线应用层协议对比研究

以下从信息标识符分配系统、过程数据交换的特点和点对点通信信道的三个方面的主要特征的对比分析不同的 CAN 应用层协议。

8.5.1 信息标识符分配系统

CANopen 提供了一个通用的标识符源，可根据设备的需求用于手动或自动的定位标识符。由系统设计者决定标识符的使用以及数据通信系统的实时行为，因此可以最大限度地使用这些可用的标识符。

DeviceNet 系统最多支持 64 个节点，每一个节点配置 3 个信息组的标识符。信息组 1 为每个设备提供一优先信息组；信息组 3 提供每个设备的低优先级标识符。对于少于 64B 的网络，不允许无用节点的标识符用于系统中。

SDS 的标识符用于说明设备的地址和服务，不是作为唯一的数据标签，因此要用一个额外的附加信息来唯一标识发送的数据。SDS 的完整数据标签包括了逻辑设备地址嵌入对象 ID 和属性或行为 ID。

8.5.2 过程数据交换的特点

表 8-9 列出了 CANopen、DeviceNet 和 SDS 过程数据交换的特点。

表 8-9　CANopen、DeviceNet 和 SDS 过程数据交换主要特点

	CANopen	DeviceNet	SDS(V2.0)
通信对象名字	过程数据对象	I/O 信息	多播信号 APDU
每个设备最大通信对象数量	512 个发送 PDO 512 个接收 PDO	27 个 I/O 发送信息 1 701 个 I/O 接收信息	32 个多播信道
数据区最大长度	8B	8B 分段	6B 分段

8.5.3 点对点通信信道

表 8-10 中总结了 CANopen，DeviceNet 和 SDS 的点对点通信信道主要特征。

表 8-10 点对点通信信道主要特征

	CANopen	DeviceNet	SDS(V2.0)
名字	服务数据通道	显示信息	点对点通道
最大信道数目	每个设备 128 个客户 SDO 128 个服务器 SDO	27 个显示发送信息 1 701 个显示接收信息 （每设备）	每个嵌入对象 4 个信道（每个逻辑设备 32 个嵌入对象）
协议	<5B： 响应 不分段 >4B： 分段发送（每段 7B） 每帧都响应 任何长度 （CAL 复合域协议）	<7B： 响应 不分段 >6B： 分段发送（每段 6B） 每帧都响应 任何长度	<6B： 响应 不分段 >5B： 分段发送（每段 3B） 接收到整个数据区后响应 最大 255B
建立连接	由 SDO 动态建立 默认的预定义连接	由不连接信息管理器动态建立 只有组 2 设备：由预定义连接组分配连接信息	由连接信息管理器动态建立 连接组的主/从组
连接服务或参数	初始化，退出 上载/下载 段/域	打开/关闭 对象的创建、配置、启动、停止、复位	打开/关闭 读、写、事件、行为
	已编址对象目录项的索引和子索引	对象属性访问路径，服务参数	信道号码/属性/行为/事件标识符

8.6 本章小结

随着 CAN 总线的广泛应用，CAN 总线的应用层协议也逐渐成为工程师们设计 CAN 总线系统所关注的焦点之一，目前国内全面介绍这类协议的书籍还较少，使用 CAN 总线应用层协议的工程师也还不多。本章在参考这些 CAN 总线应用层协议技术规范基础上，简单介绍了几种常用 CAN 总线应用层协议，供大家参考。

思考题与习题

8.1 简述 CANopen 协议的主要技术特点。

8.2 简述 CANopen 协议的 3 种工作模式的优缺点。

8.3 结合 CANopen 的典型应用案例，分析 CANopen 的网络结构。

8.4 简述 DeviceNet 协议的主要技术特点。

8.5 结合 DeviceNet 的典型应用案例，分析 DeviceNet 的网络结构。

8.6 对几种典型 CAN 总线的应用层协议，进行对比分析。

第 9 章　CAN 总线的工程应用案例

CAN 的应用领域广泛，本章结合作者和作者同事们的科研工作，对 CAN 总线在各种工程实际中的应用进行了综述。

本章要点

- CAN 总线在汽车电子系统中的应用；
- CAN 总线在工程机械中的应用；
- CAN 总线在轮轨铁路系统中的典型应用；
- CAN 总线在中低速列车状态监控与故障诊断中的应用；
- CAN 总线在高速磁浮列车状态监测与故障诊断中的应用；
- CAN 总线在工业控制领域中的应用。

9.1　CAN 总线在汽车电子系统中的应用

9.1.1　汽车 CAN 总线技术方案

上世纪 80 年代以来，随着集成电路和单片机在汽车工业的广泛应用，汽车电子控制单元越来越多，例如电子燃油喷射装置、防抱死制动装置（ABS）、安全气囊装置、电控门窗装置和主动悬架等等。在这种情况下，如果仍采用常规的布线方式，将导致车上电线数目急剧增加，一辆采用传统布线方法的高档汽车，其导线长度可达 2 000m，电气节点达 1 500 个，电线的重量可以达到 40～60kg。另外，电控系统的增加虽然提高了汽车的安全性、经济性和舒适性，但随之增加的复杂电路也降低了车辆的可靠性，增加了维修的难度。为此，改革汽车电气技术的呼声日益高涨，汽车用控制器局域网络 CAN 总线也就应运而生。

自 1989 年后，支持 CAN 总线标准的公司越来越多，其中，汽车公司有奔驰、大众、宝马、保时捷、劳斯莱斯、美洲豹等，电子公司有英特尔、摩托罗拉、菲利普、Microchip、西门子等。标准化组织 SAE 和 JSAE 也都支持 CAN 总线标准。总之，汽车总线技术的普及是汽车发展的一个必然趋势。

整辆汽车的通信网络拓扑结构如图 9-1 所示，图中的汽车计算机控制系统中的所有这些子控制系统通过 CAN 总线构成一个实时控制系统网络，各控制单元的指令发出去之后，必须保证在一定时间内得到响应，要不然就有可能发生重大事故，这就要求汽车上的 CAN 通信网络有较高的波特率和可靠性。

汽车在实际运行过程中，众多节点之间需要进行大量的实时数据交换。若整辆汽车的所有节点都挂在一个 CAN 网络上，这么多节点通过一条 CAN 总线进行通信，信息管理配置稍有不当，就很容易出现总线负荷过大，将导致系统实时响应速度下降，这在实时系

统中是不允许的。因此在对汽车上各节点的实时性进行了分析之后，根据各节点对实时性的要求，设计高、低速两个速率不同的 CAN 通信网络。将实时性要求严格、可靠性要求高的节点组成高速 CAN 通信网络，将其他实时性要求相对较低的节点组成低速 CAN 通信网络，并架设网关将这两个速率不同的 CAN 通信网络连接起来，实现全部节点之间的数据共享。

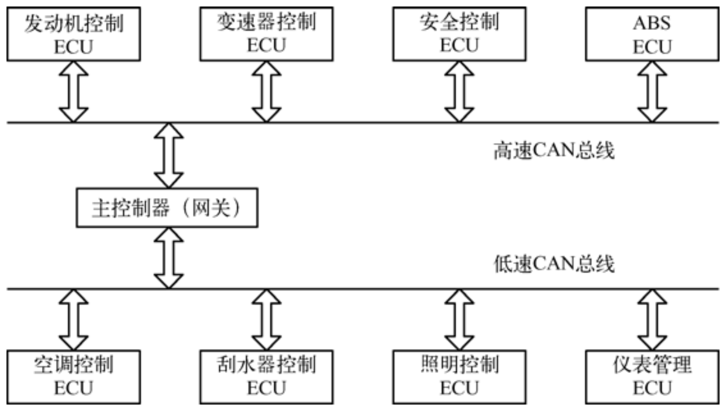


图 9-1　汽车 CAN 网络拓扑结构

图 9-2 为某型汽车 CAN 总线具体拓扑图。

图 9-2 中的发动机控制、变速器控制、安全控制、ABS 等控制单元节点是现代汽车动作的核心部件，对时间响应要求严格，因而在本设计中采用传输速率为 500kbit/s 的高速 CAN 通信网络。空调控制、刮水器控制、照明控制和仪表管理控制等相对来说对实时性的要求较低，采用传输速率小于 125kbit/s 的 CAN 通信网络，主控制器跨接高、低速两条总线，与各节点进行数据交换，兼起网关的作用，实现网络互连。

9.1.2　基于 CAN 总线的汽车远程故障诊断

随着汽车工业的发展，现代电子控制技术已渗透到汽车的各个组成部分，汽车的结构变得越来越复杂，自动化程度也越来越高，能跟踪和掌握汽车领域高新技术的维修技师和专家也必然越来越缺乏。而 Internet 随着全球信息化进程的推进得到了飞速的发展，这就为汽车维修行业间的资源共享，信息交流提供了快捷和自由的途径。另外，随着 CAN 总线技术在汽车领域的广泛应用，汽车的各种状态监控实现了网络化和数字化，这一切使得建立一个基于计算机网络的开放性的汽车远程故障诊断系统成为可能。

但由于汽车位置的不确定性，不可能通过有线的方式接到 Internet 上，而 GPRS 作为一种比较成熟的无线数据传输技术，恰好可以弥补上述缺点。通过在车载嵌入式计算机上加装 GPRS 模块，就可以实现和 Internet 的无线连接，从而为汽车的远程故障诊断系统提供了最基本的技术保证，在远程故障诊断系统中，网络化的监测系统是关键。

1．汽车远程故障诊断系统的总体结构

汽车故障诊断是指在汽车不解体的条件下，确定汽车的技术状况，查明故障部位及故障原因的汽车技术。由于是在不解体的条件下，能直接测量的结构参数（如磨损量、间隙

等）变化极少，所以在汽车诊断时，需采用一些能够反映汽车运行状况的指标，这些指标就是检测、诊断参数，如发动机转速、气缸压力、发动机功率、发动机进气温度、进气管压力等等。诊断参数有两种形式：数值参数和状态参数。数值参数是有一定单位、一定变化范围的参数，它通常反映出汽车工作中各部件的工作电压、压力、温度、时间、速度等；状态参数是那些只有两种工作状态的参数，如开或关、闭合或断开、高或低，它通常表示装置中的开关和电磁阀等元件的工作状态。要获得这些诊断参数，可以利用汽车工业已经发展成熟的汽车电子控制技术和车内通信技术，车载终端可以通过与汽车内部各功能模块的通信来获取这些参数。

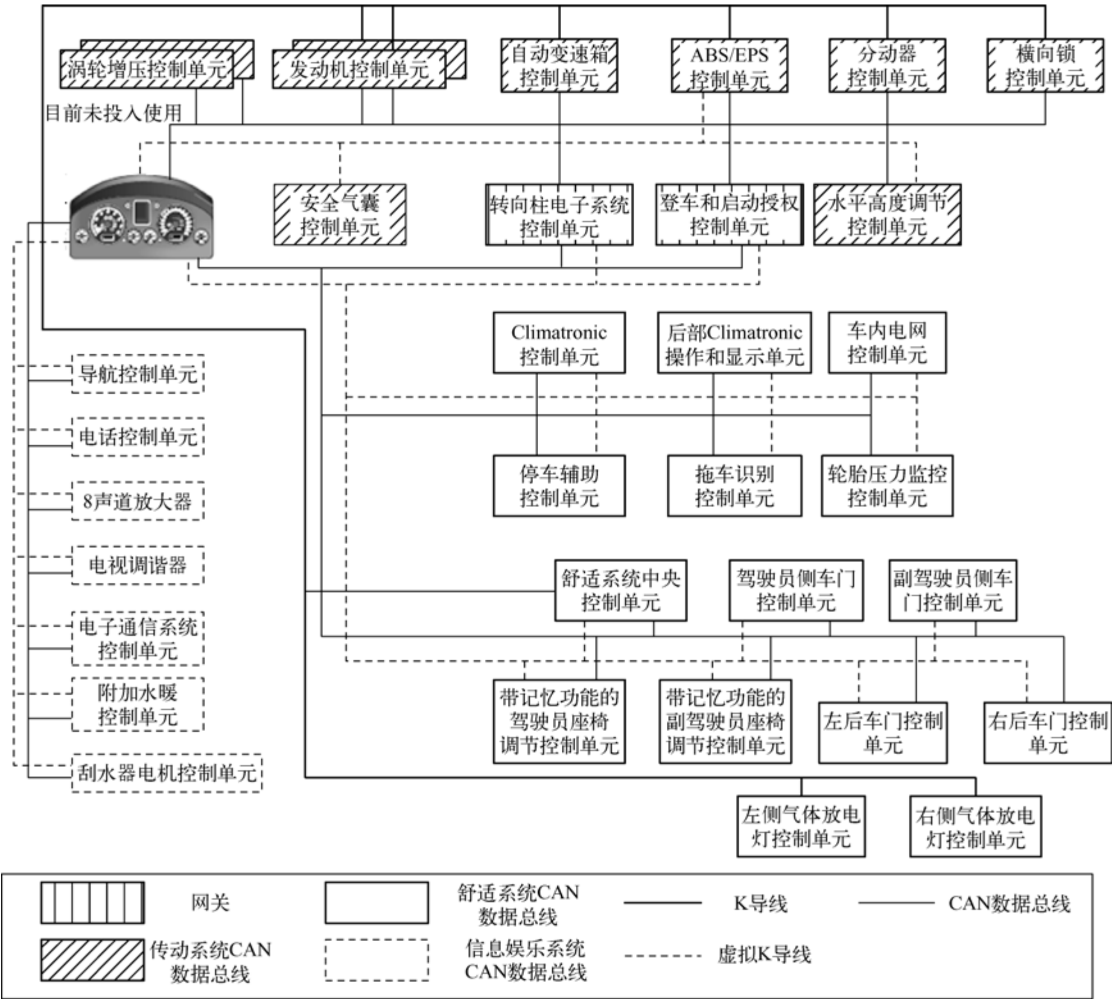


图 9-2　汽车 CAN 总线拓扑图

汽车远程故障诊断系统工作过程为用户通过车载终端对汽车上的控制模块进行数据采集和状态监测后向远程服务中心发出远程诊断请求；服务中心经权限检验后，对用户请求作出响应，启动相应模块功能，开始诊断工作，并借助网络与用户进行实时的信息交互传递。

为了详细说明诊断系统的工作过程，其总体结构还可按地理位置不同划分为三大块：

（1）车载终端

车载终端，即车载嵌入式系统，需要完成数据采集与处理、状态监测和将诊断数据打包并发送到远程诊断中心三个任务。

（2）传输通道

传输通道是异地用户与诊断中心之间的连接桥梁。由于采用的是无线网络进行数据的传输，而且目前无线网络的带宽有限，所以在数据传输时需要考虑减少图片和视频信息在网上的传输，以提高传输效率。

（3）诊断服务中心

诊断服务中心由远程诊断服务器、数据库服务器组成。用户的故障诊断请求通过网络传输到诊断中心的诊断服务器，服务器内部控制模块对诊断请求指令进行译码，并执行相应的诊断程序，同时请求数据库服务器提供数据支持。诊断完毕，将诊断结果返回到车载终端。

2．远程数据传输方案

（1）数据传输方式的选择

无线移动通信需要公共网络的参与和电话电信局或其他公共服务，目前应用较多的数据传输方式有 GPRS 方式和 CDMA 方式。

通用分组无线业务（General Packet Radio Service，GPRS）是一项以分组交换为基础的无线高速数据传输技术。GPRS 是在现有的 GSM 网络基础上叠加的一个新的网络，GPRS 和 GSM 共用相同的基站和频谱资源，只是在现有的 GSM 网络基础上增加了一些硬件设备和软件升级，是 GSM 的延续。GPRS 和以往连续在频道传输的方式不同，是以封包（Packet）式来传输，因此，使用者所负担的费用是以其传输资料单位计算，并非使用其整个频道，理论上较为便宜。

码分多址（Code Division Multiple Access，CDMA）是在扩频通信技术上发展起来的一种新的无线通信技术。CDMA 网络作为公共移动通信网络，具有联通运营商的服务支持，组网方便，维护费用低，与 GPRS 一样具有实时在线、快捷登录、按流量计费的优点。

本节选用 CDMA 方式完成车辆系统与地面监控中心的数据传输。

（2）数据传输通路的建立

根据 CDMA 网络的通信机制，可通过内嵌有 CDMA 通信模块的数据传输终端来实现车辆系统与 CDMA 网络间的数据通信，进而实现与地面综合监测预测系统间的数据传输。为节省数据传输终端的开发时间，并考虑到应用系统对终端性能的高要求，可直接选用深圳市宏电技术开发有限公司开发的 H7710 DTU 作为数据终端。

DTU 完成的基本功能有：

1）能够附着在 CDMA 网络上，通过 PPP 拨号，与 CDMA 的网关 PDSN 建立链接，获得动态 IP 地址。

2）能够通过 Socket 操作，与地面综合监测系统所在的远程中心建立 TCP/UDP 连接。

3）能够按照 CDMA 网络的协议格式对传输的数据封装和拆包。

3．无线通信模块与 CAN 总线接口设计

但是 H7710 DTU 的用户数据接口为串行数据接口（RS-232 接口），而车辆系统的状态数据大多在 CAN 总线上，因此，需要制作接口转换卡实现 CAN 口到 RS-232 接口的数据格式的转换。接口转换卡的设计以单片机为核心，进行相应的接口扩展。它主要由核心模块、电源模块、

RS-232 串口扩展模块和 CAN 口扩展模块等部分组成，下面分别针对各部分进行说明：

（1）CAN 接口模块设计

CAN 口扩展模块通过 89C51 单片机的 P0 系列端口与 CAN 总线控制器 SJA1000 的 D0～D7 相连，实现单片机对 SJA1000 的控制，并采用 82C250 芯片进行 CAN 总线数据收发，采用 6N137 芯片实现光耦隔离，如图 9-3 所示。

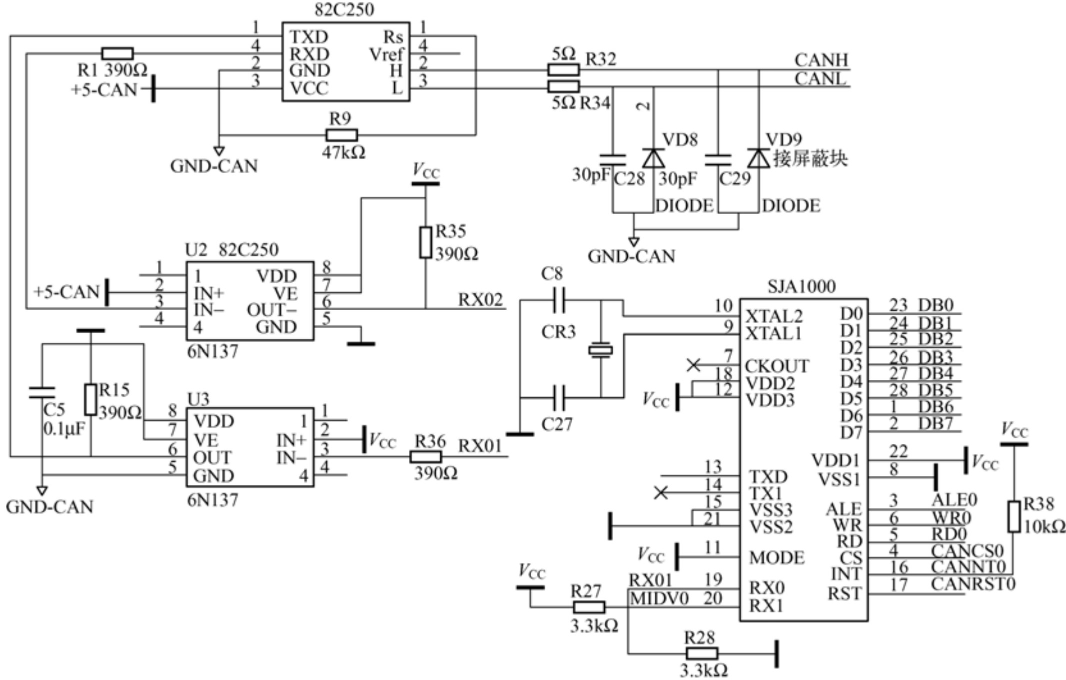


图 9-3　CAN 转 RS-232 接口卡的 CAN 口扩展模块原理图

（2）RS-232 接口扩展模块

RS-232 接口扩展模块利用 89C51 单片机的全双工串口实现串行数据收发，并通过 MAX-232 芯片实现电平转换，如图 9-4 所示。

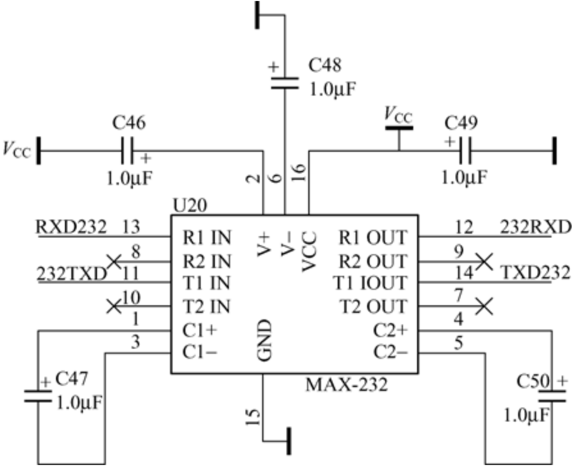


图 9-4　CAN 转 RS-232 接口卡的 RS-232 接口扩展模块原理图

在上述数据传输通路的建立过程中，车辆系统通过接口转换卡网实现 CAN 网与 DTU 间的数据传输，DTU 从 CDMA 的 PDSN 网关获取动态 IP，建立了与 Internet 的连接。此时，还需要将地面综合监测系统所在的远程中心通过无线网卡接入 Internet，建立 DTU 与远程中心的 TCP/UDP 连接。

远程中心通过无线网卡接入 Internet 时，获取的是动态公网 IP 地址，为避免每次 DTU 与远程中心建立连接时，重新设置中心的 IP，通常在远程中心服务器上运行动态域名解析软件，将中心的 IP 地址注册到域名服务器，实现 IP 地址与域名间的绑定，而 DTU 则通过向域名服务器注册获取远程中心的 IP 地址，以建立与远程中心的 TCP/UDP 连接。即可完成整个数据传输通路的建立。

9.2 CAN 总线在工程机械中的应用

随着嵌入式技术、控制网络和网络控制技术应用到工程机械控制系统中，工程机械的性能和集成度得到了很大的提高，操作更便利、安全性更高。过去，工程机械大都采用集中控制方式的方案，多个控制单元并行工作，各控制单元之间的信息交流越来越复杂，必然导致了更多的信号连接线，使控制系统安装、维护繁琐，运行的可靠性、应用的灵活性降低，维修难度增大。为提高系统的大量数据信息和控制信号在不同的控制单元中共享或实时交换，工程机械控制中传统的集中控制式系统已不能满足现代工程机械对控制系统的要求。如何提高系统的性能，开发通信应用的灵活性和方便性，降低使用和维护的成本是现代工程机械必须解决的问题。CAN 总线在工程机械控制系统中的应用，有效解决了上述问题。下面以起重机和混凝土摊铺机为例，介绍 CAN 总线在工程机械中的应用情况。

9.2.1 CAN 总线在汽车起重机控制系统中的应用

作为重要的工程施工机械，汽车起重机日趋向大吨位、高效率、自动化、智能控制及多用途方向发展。同时，用户对其工作性能、操作舒适性、安全可靠性、设备故障诊断能力以及液压系统的流量控制和分配等方面的要求也越来越高。而且，在大型汽车起重机的控制系统中，包含多个部件和分系统的电子控制单元、信号采集单元、显示单元的信号采集和传输，为提高数据的传输效率、数据和信息的共享，需要采用以 CAN 总线为部件间联络纽带的起重机网络控制系统。

1．系统主要控制对象

本节介绍的起重机为汽车底盘全液压伸缩臂起重机，额定最大起重量为 100t，5 段可伸缩主臂，1 节基础臂和 4 节伸缩臂，主臂伸缩动作的实现由 2 个双作用的油缸完成。起重机带可外伸缩支腿，由支腿控制面板和调平；液压系统为比例泵加新型流量分配的变量系统；上车独立发动机，自装卸组合式平衡重。上车起重功能的完成需 3 台油泵的工作来实现，2 台变量油泵的驱动实现臂架的升缩、变幅、主卷扬和副卷扬动作，另外 1 台油泵驱动实现转台的全 360 度回转和其他辅助动作。主要动作的操控通过 2 个电控比例手柄来实现。

2．控制系统的网络拓扑

根据起重机完成功能的要求，整车分为起重支撑控制系统、力矩限制系统、起重机操作系统、极限载荷控制系统和智能故障诊断系统。其中整车控制系统 CAN 网络由 14 个节点构

成，上车 8 个节点，底盘 6 个节点，如图 9−5 所示。

其中上车 8 个节点是：起重控制单元、力矩控制单元、集中显示单元、长角传感单元、回转角度单元、上车发动机 ECU，操作手柄 1，操作手柄 2；底盘 6 个节点是：支撑控制单元、支腿操作单元 1、支腿操作单元 2、倾斜角度单元、下车发动机 ECU 和变速器 ECU。为有效避免总线冲突，采用对总线分段，双 CAN 总线协议结构，这样既能提高系统的快速响应能力，也可有效避免由于 1 条总线太长引起的系统不稳定性。主总线符合 CAN 规范的基础上发展起来的应用层 CANOpen 协议，子总线符合 CAN2.0B 标准。其中起重控制单元和支撑控制单元具备双 CAN 网络接口（CAN2.0B 和 CANOpen 协议），既是网络节点又充当网桥的作用。

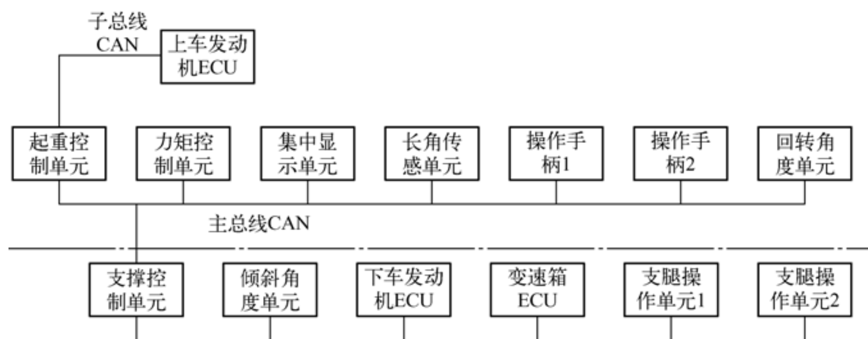


图 9−5　CAN 总线网络拓扑图

9.2.2　CAN 总线在混凝土摊铺机控制系统中的应用

混凝土摊铺机作为一种用来铺设高等级公路路面的专用设备，该设备技术性能的高低直接影响着路面的摊铺质量。为了进一步提高路面的摊铺精度，降低成本，提高工效，简化结构，国外许多公司对摊铺机的结构和振动找平等控制技术进行不断的改进和提高。本文以混凝土摊铺机自动找平控制系统为对象，主要介绍 CAN 总线在该控制系统中的应用情况。

摊铺机自动找平控制系统的工作原理是纵、横坡传感器通过 CAN 总线把检测到的路面不平整度信号送至中央控制器，然后中央控制器把得到的误差信号进行处理，再将处理过的信号输出直接驱动电磁阀带动液压油缸升降，从而消除路面的不平整度。所以根据摊铺机自动找平的工作原理，并根据实时控制的要求，确定控制器的基本组成方案。根据摊铺机自动找平控制系统总体设计，整个控制自动找平系统电气结构中共包含以下几个组成部分，如图 9−6 所示。

其中显示控制器安装在操作室内，显示每个传感器及相应控制器的参数和信息，通过按钮操作相应的控制器；横坡传感器用于检测横向倾斜坡度值，并通过 CAN 总线传送到横坡控制器及显示控制器；横坡控制器根据横坡传感器检测值、本地控制参数设定或显示控制命令对横坡执行部件进行调节，使得熨平板向倾斜的反方向运动，减小横向的倾角角度，恢复熨平板的平衡位置；纵坡传感器用于检测纵向平整度，并通过 CAN 总线传送到纵坡控制器及显示控制器；纵坡控制器根据纵坡传感器检测值、本地控制参数设定或显示控制器控制命令对纵坡执行部件进行调节，使得熨平板向高低变化的反方向运动，恢复熨平板的设定高度值，连接电缆实现整个控制系统 CAN 通信、电源、执行部件之间的连接。

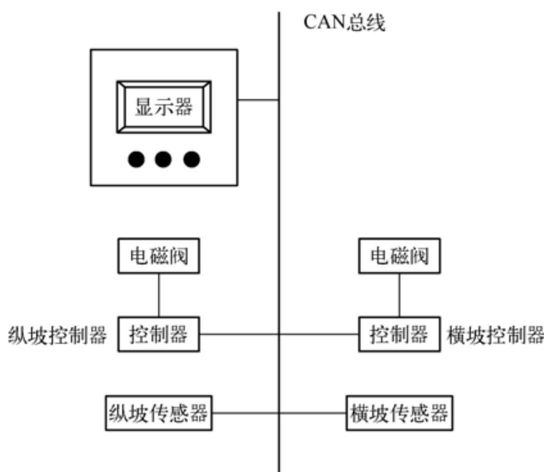


图 9-6　摊铺机自动找平控制系统总体方案示意图

9.3　CAN 总线在轨道交通系统中的应用

随着嵌入式微机控制技术和现场总线技术的发展，铁路列车的控制已从集中型的直接数字控制系统发展成为基于网络的分布式控制。IEC-61375——列车通信网络（TCN）标准是 IEC 联合 UIC 经过十年的工作采用了一个用于规范列车车载设备数据通信的标准，列车通信网络分为用于连接各节可动态编组的车辆的列车级通信网络 WTB 和用于连接车辆内固定设备的车辆通信网络 MVB。其中，MVB 适合用作车辆总线，对于固定编组的列车，MVB 也可以用作列车总线。在 WTB/MVB 总线系统中，采用主从方式下确定性的介质访问控制，所有设备只能在受控的确定时间访问介质，这样可以有固定的响应时间，但代价是需要发轮询帧，总线利用率较低，适合用于响应时间要求严格的情况下。WTB/MVB 总线系统在可靠性、可管理性、介质访问控制方法、寻址方法等方面有一定优势，但设备价格较高。

与 MVB 总线相比，CAN 总线采用改进的 CSMA（载波监听多路访问），虽然加入优先级，但不能保证实时数据在任何网络负载情况下都能满足确定的响应时间限制，因此不适合在响应时间要求严格的情况下采用，但是它的总线利用率在轻负载时较高，CAN 总线在汽车领域运用已十分成熟，其设备及软件系统具有高可靠性和高性价比的优势，因此 CAN 总线适合做通信子网络。CAN 总线与 RS-485 总线相比有很大的优势，目前地铁车辆中如运行控制、制动控制、车门控制单元、空调控制单元等控制系统使用 RS-485 总线场合的均可用 CAN 总线进行系统升级。

9.3.1　CAN 总线在轨道交通运行控制系统中的应用

列车运行控制系统是保障和监督列车按照一定的运行策略安全、高效地运行，在地铁和轻轨中称列车自动控制系统（Automatic Train Control system，ATC）。MOCS 通常完成以下三部分功能：列车自动防护（Automatic Train Protection，ATP）功能、列车自动驾驶（Automatic Train Operation，ATO）功能和列车自动监控（Automatic Train Supervision，ATS）功能。其中，ATP 用于为列车的正常运行设定一个安全边界，列车的运行状态在该边界值防护的范围以内

被认为是安全的；一旦列车的运行状态被检测到超出该边界值，就被认为是不安全的，ATP系统将自动采取紧急保护措施，保证列车运行安全。ATO 主要功能是采取一系列算法，优化列车牵引/制动曲线，为列车高效运行提供依据。ATS 对整个线路或者其中一个部分的线路状态、列车运行状态进行监督，并根据列车运行图对列车的运行、停靠时间进行调整，保证列车正点运行。ATO 和 ATS 发出的控制命令，必须经过 ATP 的鉴定被确定是安全的后才可以由地面或车载设备予以执行。从本质上看，由 ATO 执行的自动驾驶过程是一个闭环反馈控制过程，车地通信系统为上述各子系统内部及子系统之间的信息交流提供数据通道，列车的测速定位是 ATP、ATO 和 ATS 三个子系统正常发挥作用的基础。

长春轻轨和唐山磁浮试验线采用了北京通号院研制的国产化 ATC 系统，从系统设备份，ATC 系统由地面控制、车地传输和车载控制 3 个部分设备构成。其中，地面控制包含：ATS、地面 ATP 子系统（含联锁）；车载控制包含：ATP、ATO 子系统。车地传输包含：轨旁发送/接收、交叉感应环线、车载发送/接收几个部分。联锁和地面 ATP 采用区域控制方式，主控设备设置在中心，车站仅设执行设备。站间闭塞由联锁系统完成。下面分别介绍该 ATC 系统中 CAN 总线的应用情况。

1．车载运行控制系统的 CAN 通信网络

ATC 系统中的车载 ATP 的主控计算机和运行状态显示屏的连接就是采用 CAN 总线完成的。其组成框图如下：车载 ATP 主机与车载 TWC 接收器、车载 TWC 发送器、车载 ATO 主机、车载 ATP 接收器、车载记录器、车载人机子系统为 CAN 通信链路，协议为 CAN2.0B 扩展帧格式，物理通信介质为屏蔽双绞线，图 9−7 为其结构示意图。

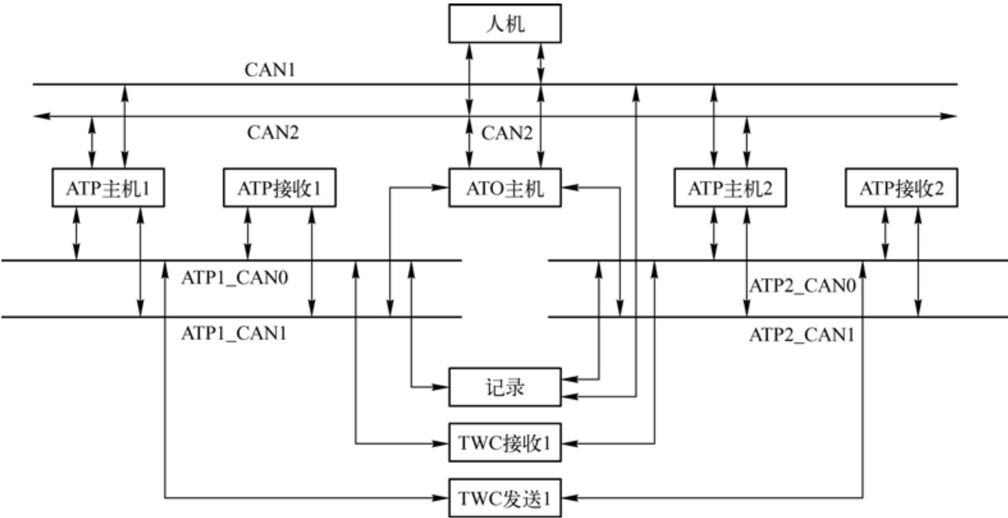


图 9−7　CAN 总线在车载 ATP 系统中的应用

2．地面区域控制中心的 CAN 通信网络

地面区域控制中心的通信单元内部也采用了 CAN 总线，作为内部设备通信总线。整个通信单元隶属于数字轨道电路部分，是区域控制中心和轨道电路（包括环线）通信的中继，接收 ATP 区域控制中心发送来的数据，分发后发向各个轨道电路（包括环线），同时接收各个轨道电路（包括环线）的状态信息及设备维护信息，编码后发送给区域控制中心。

（1）系统结构

每个轨道机柜包括一个通信单元，通信单元为双机热备结构，由主备通信板构成，通过CAN 总线分别与区域控制中心和轨道电路（包括环线）相连，进行通信，区域控制中心 CAN 总线和轨道电路 CAN 总线均为双冗余结构。区域控制中心最多管理 10 个通信单元。每个通信单元最多管理 8 段轨道电路（包括环线）。结构示意图，如图 9-8 所示。

（2）通信单元的总线结构

单一通信板为双 CAN 总线结构，CPU 同时与两路双 CAN 总线相连，总线为 CANA、CANB，轨道（或环线）总线为 CAN1、CAN2。CANA、CANB 同理于 CAN1、CAN2，两路总线均是硬件冗余。总线上同时存在数据，CPU 均接收处理，但只使用其中一条总线上的数据，当一条总线故障时，换用另一条总线上的数据，其原理相当于双机热备。CPU 对两条总线上的数据不进行比较，数据的可靠性依靠 CRC 校验来保证。

（3）通信协议

与区域控制中心通信，通信板与控制中心需要交换的数据量比较大，所以，采用的通信速率为 500kbit/s，通信周期为 250ms，协议为 CAN2.0 通信协议。与轨道电路（包括环线）通信，从轨道电路（包括环线）的传输特性及通信板与轨道电路（包括环线）交换的数据量考虑，这部分的通信速率采用 250kbit/s，通信周期为 150ms，协议为 CAN2.0A 通信协议。

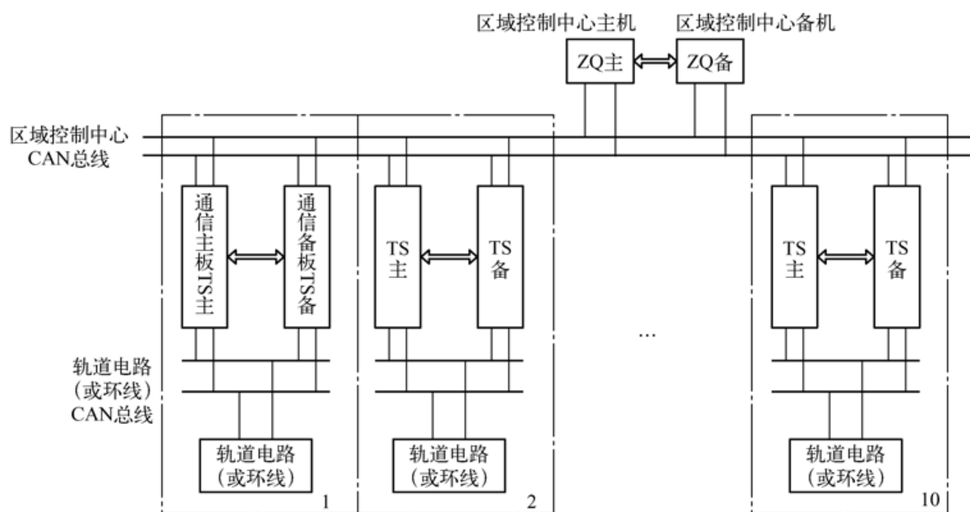


图 9-8　CAN 总线在地面区域控制中心的应用

9.3.2　CAN 总线在内燃、电力机车的运行监控记录装置中的应用

在铁路运输的发展过程中，运输安全始终是人们最关注的问题。自 1995 年以来，全国大部分的内燃、电力机车上均安装了 LKJ 系列运行监控记录装置。国际上，各铁路强国也研制了各种类型的 ATC、ATP 装置，如日本最新的数字化 ATP、法国的 TVM300、TVM430 等，目前它们都在国外高速铁路上被推广使用。与国外这些装置相比，我国监控装置采用车载微机内存运行线路参数的方式，设备简单、投资少，并有较强记录功能，但在多机冗余配置及与地面信息功能方面比国外弱。根据参考文献[41]，该运行监控记录装置也应用了 CAN 总线。

1．运行监控记录装置主要功能

监控功能：防止列车线路超速；防止列车冒进关闭的进站信号机；防止列车冒进关闭的出站信号机；防止列车溜逸；防止列车以高于规定的限制速度调车作业；按列车运行要求控制列车不超过临时限速。

记录功能：开/关机时相关参数记录、乘务员输入参数记录、运行参数记录、事故状态记录。

显示功能：显示列车运行的实际速度及限制速度（或目标速度）；显示距前方信号机距离及前方信号机种类；显示运行线路状态；显示机车优化操作曲线。

地面分析处理功能：将车载记录的列车运行数据经过翻译、整理、以直观的全程记录、运行曲线、各种报表等形式再现列车运行全过程，为机务的现代化管理及事故分析提供强有力的工具。

2．CAN 总线在该运行监控记录装置中的应用

采用双机热备冗余工作方式、双套插件、双套 CAN 总线、双套 VME 总线、模块级冗余、主备机故障自动切换。系统组成框图（见图 9-9）。

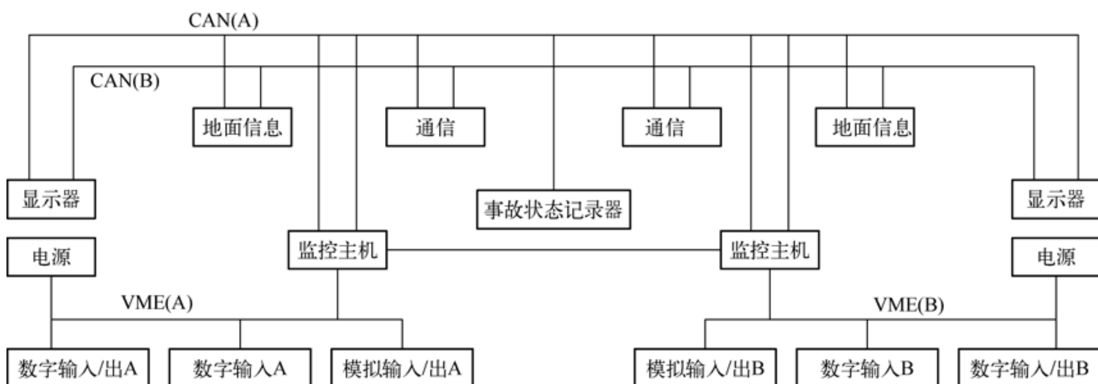


图 9-9　CAN 总线在 LKJ2000 型列车运行监控记录装置的应用

系统通信分为三级：第一级为 CPU 级，第二级为系统级，第三级为外部级。其中第一级为主机的工作机与热备机之间的通信，第二级为主机与显示器、地面信息处理模块及通信模块之间的数据通信，第三级采用通信模块扩充通信接口。

该装置采用双路 CAN 总线是为了提高可用性，即在一路 CAN 总线故障的情况下能够保证正常使用。为此，全部软件需要解决双路 CAN 总线冗余工作的问题。在软件设计中，各软件采用如下算法处理双路 CAN 总线通信。

发送的数据帧同时发送至 CANA、CANB 上，从 CANA、CANB 接收的同一内容的数据帧存入同一缓冲区，但延时一定时间后再使用。这样，在一路 CAN 总线故障，另外一路正常的情况下，总保证有一路能够正常发送收发，接收的数据进行延时后再使用，可保证从双路 CAN 总线接收的同一内容的数据帧不会被重复处理。

9.3.3　CAN 总线在轨道交通车辆制动控制系统中的应用

目前国内各城市地铁车辆所用制动控制系统，许多列车是采用了克诺尔 EP2002 制动模块，或以 EP2002 为核心模块进行二次开发而成的国产制动控制系统。EP2002 相当于将常规制动控制系统的制动电子控制单元 BECU 和制动控制单元 BCU 集成为一个模块。列车各类

EP2002 之间就是通过 CAN 总线进行通信。例如在南京地铁 2 号线制动控制系统的 CAN 总线采用两对双绞线的方式，具有良好的冗余性。单个制动模块的故障只会导致一个转向架失去制动力，对其他系统无影响，因此该制动系统具有很高的安全性。

下面具体主要以应用在唐山磁浮试验线和大连轻轨中的国产化制动系统为例，介绍 CAN 总线在其中的应用，制动系统由制动指令发生及传输系统、制动控制系统、动力制动装置、基础制动装置、风源系统、风缸、气动系统附件（如管路、截断塞门、软管、滤清器、单向阀等）组成，其系统组成，如图 9-10 所示。其中 CCU 为中央控制单元，BECU 为制动控制单元，DCU 为牵引控制模块。

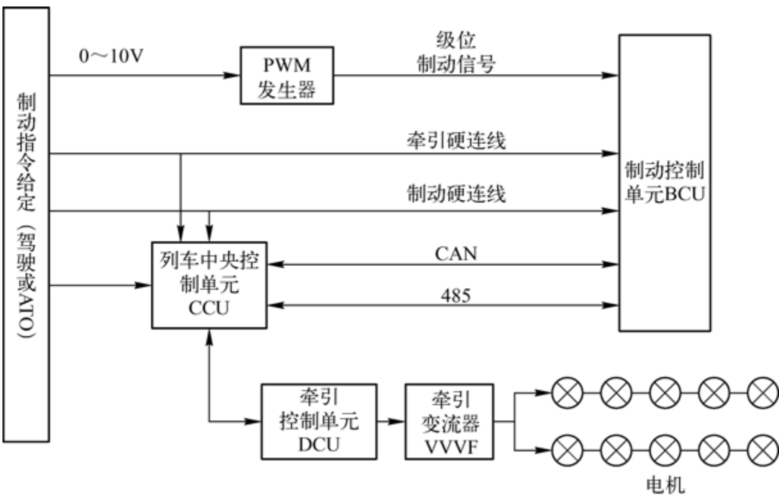


图 9-10　制动系统结构图

在该系统中制动控制主要单元 BECU 就是通过 CAN 总线与车辆的列车控制单元 CCU（VCU）进行数据通信。

常用制动时，优先使用动力制动。此时，牵引电动机变成发电机，将动能转换成电能，通过电阻逸散出去。当动力制动力不足时，空气制动可自动补偿。

制动时，驾驶员控制器发出制动指令，传到每辆车的制动控制单元 BECU。如果动力制动有效（DCU 向各车 BECU 提供动力制动有效信号），BECU 接到制动指令后，根据制动指令、车辆载荷计算出所需制动力，该制动力作为动力制动需求指令通过 CCU 提供给本车 DCU，DCU 根据该指令和自己计算的制动需求，按照取大的原则施加动力制动力，同时将实际的动力制动力反馈给本车的 BECU， BECU 将实际动力制动力和所需制动力进行比较，如果实际的动力制动力能够满足所需制动力，则车辆不施加空气制动，如果不能满足，则计算出所需制动力和动力制动力之间的差值，根据该差值施加相应的空气制动。

9.4　CAN 总线在磁浮列车状态监测与故障诊断中的应用

9.4.1　CAN 总线在中低速磁浮列车状态监测与诊断系统的应用

中低速磁浮列车车载状态监控与诊断系统包括车载状态监控系统和车载诊断系统两大部

分。车载状态监控系统主要用来实现列车车载设备的状态监测与控制功能。车载状态监控系统的主要功能有两个：一方面把由车辆底层设备产生的各个状态信号进行处理后传送给车载控制系统；另一方面是将车载控制系统发出的控制命令经处理后分配到相应车辆设备，从而控制这些设备实现其功能。

车载诊断系统主要用于对车载电气设备进行在线诊断，通过采集列车上各设备的状态信息，判断这些设备是否发生功能性故障，并采取一定的方式对这些故障进行综合性评估，根据评估结果提示驾驶员采取相应措施，保证列车的安全运行。通过诊断系统能够实现故障诊断状态信息的显示，同时还可实现部分与安全无关的控制功能。车载诊断系统主要由列车诊断计算机、显示模块、键盘和两级的传输网络以及底层的被诊断功能设备组成。

本节首先从总体上分析车载监控与诊断系统的功能要求，选择确定列车通信网络方案。

1．系统功能设计要求

磁浮列车车载状态监控与诊断是列车安全稳定运行的重要保证，其主要功能要求有：

1）能够对磁浮列车的运行进行全面统一的正确控制与监测，以保证驾驶员操纵作用，并时刻掌握列车的运行状态，确保列车的安全、快捷、舒适运行。

2）能够通过综合监测磁浮列车系统级控制设备以及列车中每节车辆的各受控设备，确保它们都能按照驾驶员操纵和行车指挥命令相互协调工作。

3）能够迅速地检测出列车控制与诊断所需的各种信息，包括过程数据、消息数据、监控数据，准确地监测和识别运行中车辆所发生的故障，提示驾驶员采取措施及时予以排除。

4）需要将运行过程中设备故障发生的时间、位置、相关参数的当前值及变化情况等信息记录储存，这些数据要求能在显示屏上显示，也可以通过其他传输装置发出，提供给地面系统做进一步分析处理，为列车检修提供信息资料，缩短维修、维护保养时间，节约列车维护成本。

5）在列车故障情况下提示运行方式，包括提出保持功能措施的建议与迅速排除故障的维修方式。

总之，磁浮列车车载状态监控与诊断系统需要实现对列车的全面运行监控，并把分布在不同车辆中的各个控制与受控设备的状态、故障等信息收集、处理分析、显示、存储起来，实现列车设备装置分布式联网，同时要求系统具有极高的可靠性。

2．列车通信网络方案

磁浮列车的状态监控与诊断系统是车载分布式的计算机网络系统。在每节车辆内通过车厢级总线将分布在同一车厢内的各个控制装置与控制设备联网；通过列车级总线把分布在不同车厢中的主控单元节点联网，直至安装在列车前后两端车上的列车监控与诊断中心，再通过端车驾驶员操纵台上的显示屏，可选择地显示列车中各受控设备的工作状态。从而实现对列车的重连控制和对全列车的综合监控与诊断作用。

列车运行控制模式由主控操纵端确定，中央控制单元根据运行控制模式对列车运行进行相应的逻辑处理和控制。列车在正常运行控制模式下，网络系统对列车运行控制进行完整的逻辑控制和监测；解锁运行控制模式下，网络系统对列车运行进行控制，但是网络系统旁路所有外围控制条件，根据输入信号直接对列车进行控制和监测；应急运行控制模式下，网络系统仅对列车运行进行监测，网络系统屏蔽所有控制功能；列车为 ATO 驾驶模式，则列车在网络和应急控制模式下均能正常运行。

列车车载控制与诊断系统结构，如图 9-11 所示，系统主要由以下部分组成：

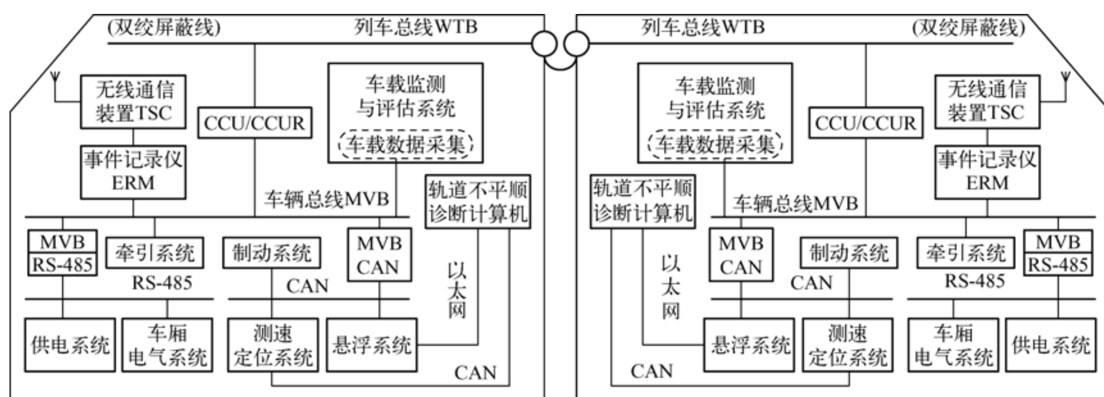


图 9-11　CMS-04 型磁浮列车控制系统结构示意图

1）车载控制器：即 CCU/CCUR，完成信息监测与处理、控制指令生成、数据存储等功能。

2）控制对象：包括低压柜、高压柜、AD110 电源、DD330 电源、辅助逆变器、悬浮控制器、牵引逆变器、制动控制系统、车门单元、空气调节器等设备。

3）控制开关：包括驾驶员钥匙、操纵手柄和控制开关，用于驾驶员操纵。

4）显示装置：包括状态监控与故障诊断评估计算机、运行控制信号屏、指示灯和模拟表，用于显示列车和设备的状态和故障诊断等数据。

5）信息通道：包括列车总线、车辆总线、列车控制线和车辆控制线，用于传递控制指令、控制参数、监测列车状态和数据。

CMS-04 的列车控制与诊断网络由以下部分组成：

1）列车总线：贯穿全列车，连接各车的 CCU/CCUR 和 VCU，用于传递列车信息。

2）CCU/CCUR 和 VCU：用于完成列车控制与车辆控制，并作为列车总线与车辆总线之间的网关。

3）车辆总线：连接车辆内部的各设备，用于传递车辆信息。

4）车载设备总线：与车辆总线相接，通过网络与 CCU/CCUR 和 VCU 交换信息，通过 RS-485 或者 CAN 总线与车辆总线交换状态信息。

9.4.2　CAN 总线在高速磁浮列车车载诊断系统的应用

1．系统功能要求

高速磁浮列车车载诊断系统主要用于对车辆电气与电子部件进行在线诊断，通过采集车辆上各部件的状态信号和部分模拟量，来判断各部件是否有功能故障。通过诊断系统还可以实现部分控制命令的输入操作和状态、诊断信息的显示，同时还可实现部分与安全无关的控制功能。

车载诊断系统主要由列车诊断计算机、列车操作计算机及其显示模块和键盘、车辆诊断计算机、两级的传输网络以及底层的被诊断功能部件组成。列车级诊断计算机对整列车的故障程度进行评估并将这些数据通过无线传输装置发送到地面，并由无线地面接收站进一步传

送到运行控制中心和维护中心等。其故障诊断评估系统从设备、车辆、列车三个层次实现故障综合评估。

另外，高速磁浮列车的牵引供电、运行控制和线路轨道系统都设有相应的状态检测、故障诊断与综合评估系统。

车载诊断系统在每节车上都有一个车辆诊断计算机（SDR），车辆内的各部件通过 CAN 总线与所在车辆的车辆诊断计算机连接从而形成一个诊断子系统。各车辆的功能部件通过其内部的信号采集装置采集控制信号和状态信号，并将这些信号通过 CAN 总线传输给所在车辆的车辆诊断计算机。采集到的状态信号在车辆诊断计算机通过与设定值的比较来判断车辆部件是否出现故障。各车辆诊断计算机、分别位于前端车与后端车的两个列车诊断计算机（FDR）和两个列车操作计算机（FPR）之间通过工业以太网相连接。每个列车诊断计算机与列车操作计算机都带有一个显示模块和一个键盘，显示模块和键盘通过 RS-232 串行总线与计算机连接，实现人机接口，可以进行各种信息的显示与操作。其中列车诊断计算机用于显示与存储各节车辆的诊断数据，列车操作计算机可以存储各节车辆间传输的操作数据。列车诊断计算机与列车操作计算机通过以太网与车载无线电设备连接；通过 TTY 串行电流环接口与车载安全计算机（VSC）相连接。

2．列车通信网络方案的确定

高速磁浮列车的车载诊断系统通信网络方案采用工业 EtherNet+CAN 通信网络系统结构，该结构也采用车厢级与列车级的分层网络结构。如图 9-12 所示，在车厢级中，监控/诊断系统与底层设备之间由 CAN 总线实现连接；在列车级中，通过标准的以太网总线结构把各个车辆之间联系起来；整体上利用嵌入式透明网关实现以太网和 CAN 总线之间的无缝连接，由此达到协议转换和信息交换的目的，实现对整列车的状态监控与诊断。

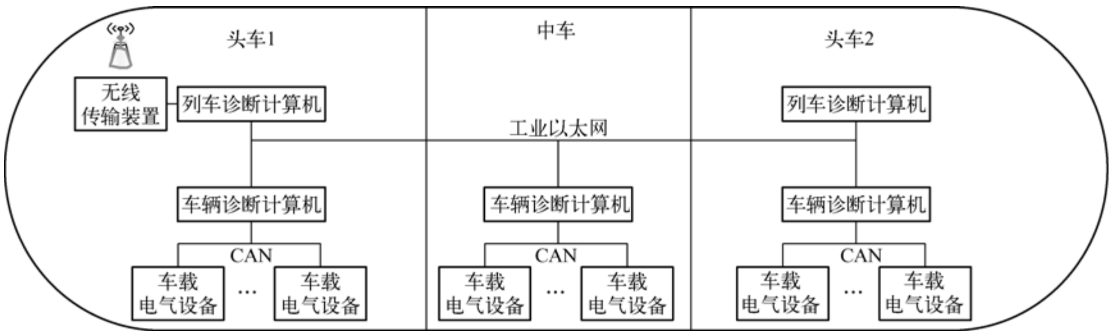


图 9-12　高速磁浮列车故障诊断系统的结构图

9.5　CAN 总线在工业控制中的典型应用

9.5.1　CAN 总线在香烟包装机、卷接机生产线上的应用

本节以某卷烟厂 CIMS 中的包装机、卷接机车间数据采集和信息处理系统为对象，采用

CAN 现场总线技术，对该厂的 18 条包装机和卷接机生厂线进行了技术改造。下面以包装机为例进行介绍。

卷烟生产的特点是工序多、流程长、工艺复杂。在国内卷烟生产企业中，制丝、卷接包等生产过程大部分已经采用国外进口的生产设备，虽然单机自动化程度比较高，但有不少设备没有数据通信功能，该卷烟厂的 GD 系列包装机就是如此，由于该厂的 GD 系列烟支包装机生产线引进较早，其电控部分全部由时序逻辑电路实现，无法与车间的 MIS 系统相连，通过在每条 GD 包装机生产线安装一台 AWS-82515 CRT 整合式工业级 PC 作为其数采站，配置有四块自制的信号接口板以及 CAN 网卡等，数采站通过四块信号接口板实时采集现场的产量、消耗、成品率、设备状态等数据，数采点并在 GD 电控柜的传感器输入点和系统输出点上，数采站还通过 CAN 网卡与车间管理机进行数据交换。

1．数采站具体功能如下：

1）实时采集 GD 机的主要产耗数据（包括正品烟产量、正品烟包数、商标纸消耗、铝箔纸消耗、条合片消耗）。

2）实时采集 GD 机的设备状态数据（包括有效作业率、成品率、总开机时间、正常运转时间、停机时间、停机次数）。

3）实时采集 GD 机的剔除器动作频度和总量（包括烟支剔除、铝箔剔除、三轮剔除、商标纸剔除、六轮剔除、八轮剔除、双包剔除、条包剔除）。

4）实时采集 GD 机的故障停机原因和停机时间（包括所有故障停机原因信号）。

5）实时采集 GD 机的剔除原因数据。

6）实时接收并显示来自上级管理部门的通知或命令。

7）数采站可以向车间维修组或发料房发出维修和需料请求以及质检数据。

8）数采站可实时显示包装机的各种现场数据，以设备流程图或模拟图等形式实时显示设备的运行状况、停机原因及位置，实时显示各机台产量对比情况。

9）数采站可以调看本机台前两班的有关历史数据。

10）采样间隔为 2 ms，数采误差＜5‰。

2．监控网络的结构

包装机数采系统网络结构如图 9-13 所示。由于这些 GD 机数采系统都是建立在工业 PC 基础上，所以要求相应的 CAN 通信接口卡为 PC 总线适配卡。经过比较，采用智能 PCCAN 适配卡。该卡上有高性能的嵌入式微处理器 80C188，极大地减轻了主机 PC 的通信负担，而且可以运行用户复杂的通信任务，卡上有 2KB 的高速双口 RAM 直接映射到主机内存空间，操作时，用户通过软硬件设置将卡上的双端口 RAM 映射成 PC 的物理内存，这样用户收发数据就相当于直接向内存读写数据，从而极大地提高了通信卡和 PC 总线的数据交换速率。

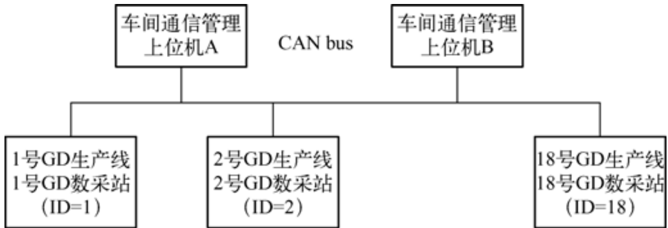


图 9-13　CAN 网络结构示意图

实现 CAN 与主机 PC 的高速数据交换，具体安装时，只需将 PCCAN 直接插入到 PC 机的 ISA 总线扩展槽上，再将卡上的 DB-9 插座按 CiA 标准与双绞线相连即可。

课题组还将 CAN 总线技术成功应用到烟厂的卷接机生产线的数采系统中，目前全厂共有 4 条 CAN 网，50 多个 CAN 节点投入使用，整个系统从 1999 年初试运行，一直稳定可靠。

9.5.2　CAN 总线在隧道窑控制系统中的应用

隧道窑是一种连续式窑炉，由预热区、高温区、急冷区和缓冷区组成，主要用于建筑陶瓷、日用陶瓷等烧制。隧道窑的控制涉及传动控制，温度、压力的检测控制以及其他控制。

1．系统功能要求

1）温度控制区 8 组。

2）温度显示区 10 组。

3）热电偶：18 支。其中：K 分度 11 支，S 分度 7 支。

4）风机类型：

排烟风机：15kW，一开一备，星−三角启动；

助燃风机：15kW，一开一备，要星−三角启动；

急冷风机：11kW，一开一备，变频控制；

抽热风机：15kW，一开一备，星−三角启动；

快冷风机：7.5kW，一开一备，直接启动；

轴流风机：0.18kW，8 台，直接启动。

5）安全联锁，采用开关量互锁：必须是硬件互锁和软件互锁，在设计时进一步落实工艺互锁要求。

6）现场设人机界面，所有操作均可在人机界面上完成；面板按钮与人机界面按钮互为备份。

7）有关控制信息能传到计算机，计算机只作信息管理用，不参与控制，计算机画面应含有：现场模拟图、温度实时曲线图、控制设定曲线图、历史趋势曲线图、并完成报表打印功能。

2．控制设备选型

据功能要求，进行系统分析，共有温度采集 8 点，开关量输入 49 点，模拟量输入 8 点，开关量输出 34 点，模拟量输出 8 点，系统控制平台采用了湖北黄石科威自控有限公司制造的 EC 系列 PLC 温度采集模块（EASY-1600N）1 台，温度控制（EC-08M08R-04N04B）2 台，风机控制部分采用（EC-16M16R）2 台，另需做主站的 PLC（EC-16M16R）1 台，用于控制拖车部分的 PLC（EC-16M16R）4 台，显示控制的人机界面 1 台（10.4 英寸，真彩色），用于监控的普通计算机 1 台。其他电器部分：变频器、开关电源、空气开关、接触器、继电器、按钮、指示灯、报警器、柜体等。控制系统结构图如图 9−14 所示。

图中，各模块主要性能参数如下：

EASY-1600N：配置 16 路温度输入，含与 CAN 网连接的现场总线接口，可与各采集模块、Easy 系列的 PLC 互连组成网络。

EC-08M08R-04N04B：配置包括 8 点开关量输入、8 点开关量输出，4 路模拟量输入、4

路模拟量输出，CAN 网连接的现场总线接口，可与各采集模块、PLC 互连。与通用人机界面相连的 RS0 串口，便于现场监视和操作；有与计算机相连的 RS1 串口，便于计算机记录管理。

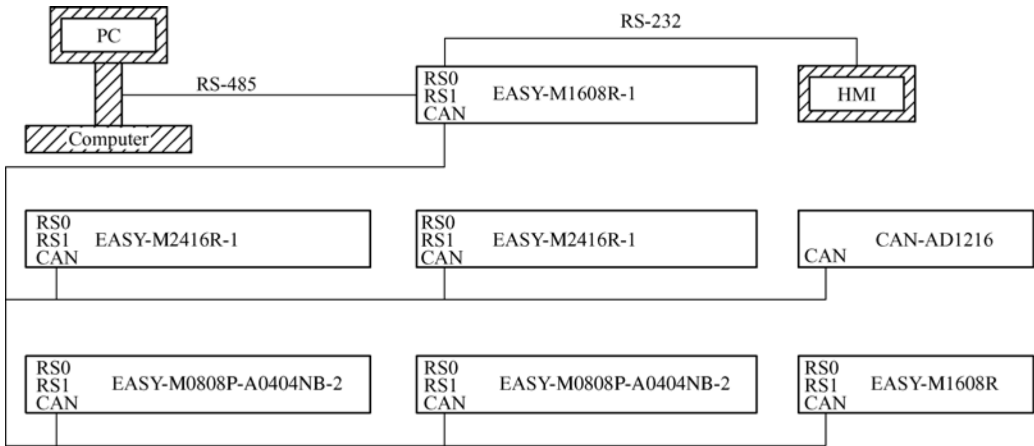


图 9-14　CAN 总线在隧道窑控制系统中的应用

Easy-M2416R：配置了 24 点开关量输入、16 点开关量输出，与 CAN 网连接的现场总线接口，可与各采集模块、PLC 互连。有与通用人机界面相连的 RS0 串口，便于现场监视和操作，有与计算机相连的 RS1 串口，便于计算机记录管理。

Easy-M1608R：配置了 16 点开关量输入、8 点开关量输出，有与 CAN 网连接的现场总线接口，以便于各采集模块相连，有与通用人机界面相连的 RS0 串口，便于现场监视和操作。有与计算机相连的 RS1 串口，便于计算机记录管理。

3．CAN 网络配置

上述控制设备组成一个 CAN 总线网络，整个 CAN 总线网络互联由 CANSET 软件来实现，它是图形化界面的软件，设置起来十分简便。设置内容包括：网络设备总数、网络设备地址、网络通信数据的内容等，以保证网络资源的合理分配、网络数据通信速度等。它还能根据用户的需要，灵活设置每个设备的任务级别。

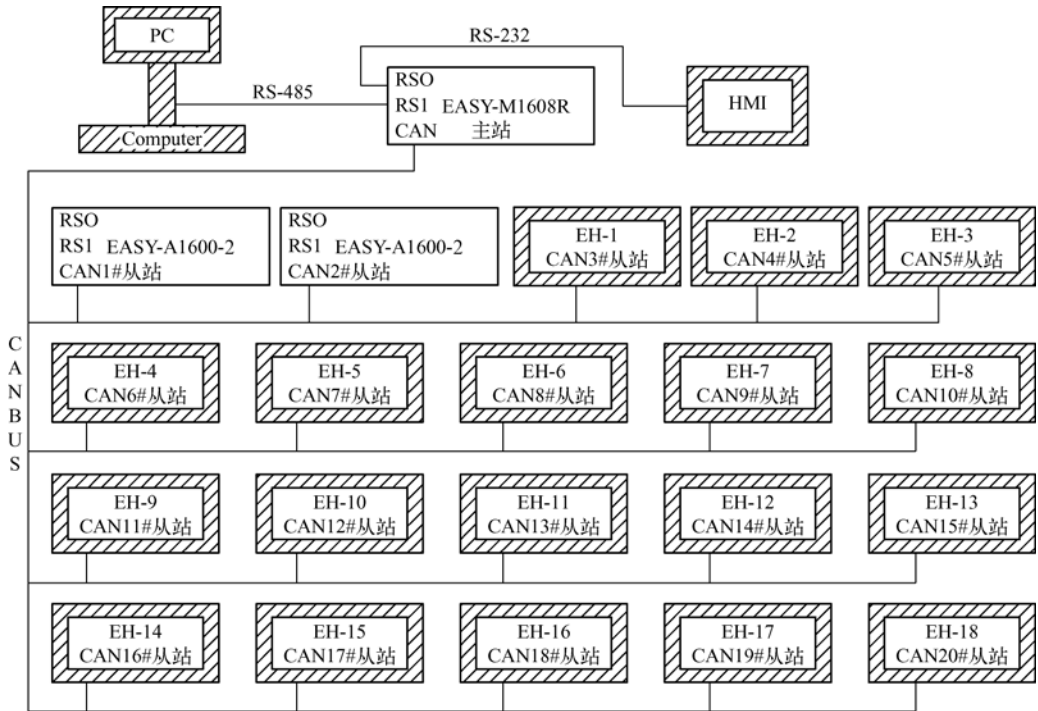
该控制系统采用 EASY 系列 PLC，通过 CAN 总线连接各个控制设备，硬件结构简单，成本低廉，响应速度快，性价比高，还可以根据实际需要很方便地进行扩展，有极强的灵活性和适应性。现场使用表明，其性能稳定，运行可靠。

9.5.3　CAN 总线在网带窑控制系统中的应用

电烧网带窑是一种底烧式及上烧式网带传动连续生产的热工设备，在网带下面设置了多组电阻丝进行加热，窑内高温气流进行反复搅拌，由无级调速电动机，带动减速机及主动滚筒，使耐热不锈钢网带运动，从而带动产品经过预热、高温、冷却段，完成烧制过程的一种窑炉。

EH 电加热控制器是一种面向工业电加热行业的专用控制设备，该控制器集功率晶闸管、晶闸管移相（或过零）触发、微电脑控温（或控压、控流）板于一体，具有 CAN 总线通信、电流电压反馈、超温、过流报警等多项功能，是解决高精度电加热温度控制的理

想装置。控制系统中的 EASY 系列嵌入式 PLC 编程环境与三菱 FX2n 兼容，除具有通用 PLC 功能之外，还可与多家 HMI 相连，能方便地实现 CAN、485 通信，可组成多层网络如图 9-15 所示。



9-15　CAN 总线在网带窑控制系统中的应用

9.5.4　CAN 总线在双变频卷染机控制系统中的应用

在染织行业中，卷染过程在整个布匹加工过程中起着重大的作用，这一过程主要是通过将布坯在卷染机内进行多次往复卷染，去掉布坯上的染浆或杂丝，从而对布坯进行均匀染色等。卷染机适合于目前市场对多品种小批量织物的染色需求，可间歇式生产，发展前景看好，应用越来越广泛。

1. 系统功能要求

卷染机控制方面要求具备自动记道、自动计数、自动换向、自动掉头、自动停车等功能，在整个工艺过程中，要求保证布匹的张力和线速度恒定是最重要的。国内较为传统的卷染机大部分采用双直流电动机控制，只能达到近似的恒张力控制效果，也有采用单变频器的卷染机，放卷采用异步电动机直流制动的方式，收放卷用接触器在变频器和直流制动之间进行切换，以上这些方案，分析其原理，都是在较大误差下的一种近似结果，因此控制效果不尽如人意。

进口的高档卷染机，有的是采用液压控制，有的是采用伺服控制，也有的是用价格昂贵的工程型变频器来实现，效果都较为理想，但是对于国内的大部分用户来说，成本压力很大。

传统的进口液压控制系统也有采用液压泵与比例节流阀组成的泵源，可以实现液压马达的无级调速，但该类系统维护检修麻烦，而且价格也特别昂贵。本节介绍的案例，是采用 EC 系列张力专用控制器 PLC 加科威矢量变频器和 HITECH 公司的人机界面解决方案。

2．控制网络结构

系统采用 1 台 EC-16M12R-04K02B-6P 张力专用 PLC 和 2 台矢量变频器，通过 CAN 总线组成主从站网络，RS-485 组成通信网，从而完成设备启停，温度、张力控制以及速度控制，如图 9-16 所示。

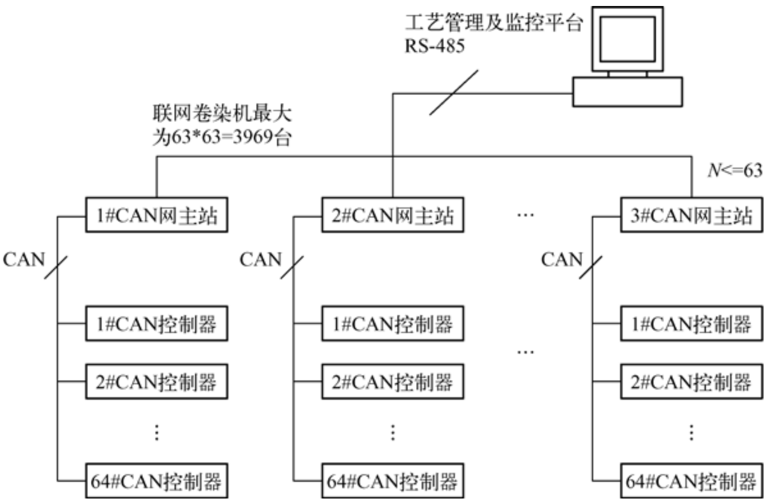


图 9-16　CAN 总线在双变频卷染机控制系统中的应用

人机界面通过 RS-232 与主站 PLC 的 RS0 口连接，所有手动操作、参数设定、工况显示均通过人机界面实现。在工厂可以配置一台计算机作为服务器，通过 RS-232 与主站 PLC 的 RS1 口连接，监视所有机器设备运行状态并记录运行数据。其他授权网络用户也可以通过访问服务器方便地察看设备运行情况或查阅设备历史数据。

每一台机器设备配一台控制器，因此网络结构可以连接 3969 台卷染机。工艺管理和设备状态监控均可在网络平台上执行。

CAN 总线功能包括监视每台机器设备的工作状况，下载相关工艺参数。每台机器所配的控制器，均已有现场总线接口，无需再配置通信模块；每台机器上的控制器均是现场总线从站，从站地址和通信速率可设，但通信速率必须与主站一致。组建网络时，63 台设备配一个 CAN 主站，CAN 主站可选通用 EC-08M08R 即可；网络增加时，CAN 主站数目增加；多个 CAN 子网并行工作，通信效率与一个子网相同。

9.5.5　CAN 总线在圆网印花机控制系统中的应用

传统圆网印花机的印花控制主要是靠机械传动来实现导带与网的同步，机械传动误差比较大，又容易磨损且维修极其不便，嵌入式 PLC 产品——运动控制器在圆网印花机控制系统中的应用，利用运动控制器的强大功能，实现主电动机与网头分电动机间的同步，采用 CAN 总线将所有状态通过人机界面展现给用户。用户可以根据工艺要求用梯形图语言编程，灵活

达到控制目的，从根本上彻底解决了上述问题。各个运动控制器之间通过 CAN 总线相连，如图 9-17 所示。

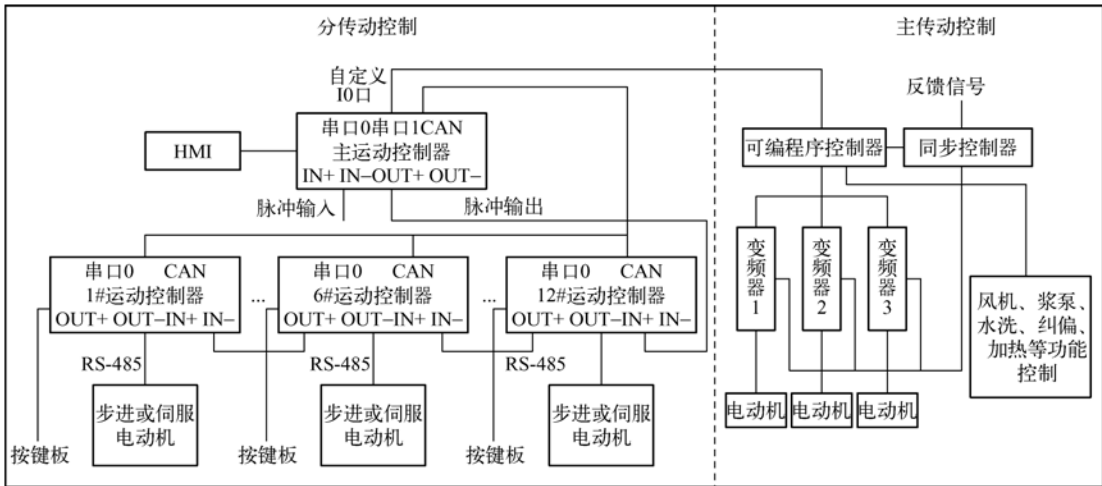


图 9-17　CAN 总线在在圆网印花机控制系统中的应用

圆网印花机控制系统分主传动和分传动两大部分：主传动主要是控制进布电动机、超喂电动机、印花电动机、烘房电动机、立柱烘电动机和落布电动机几个单元之间的同步。分传动则是实现印花主电动机与网头分电动机间的同步，精度要求高。

9.6　本章小结

CAN 总线应用广泛，本章介绍了作者和作者同事们应用 CAN 总线负责或参加完成的部分工程项目。为了尽可能给读者呈现更多的 CAN 总线在工业控制领域的应用例子。在作者的邀请下，湖北黄石科威自控有限公司龚云生同志为本书第 9.5 节的案例编写提供了许多 CAN 总线在工业控制领域的应用实例，在此一并表示感谢。

思考题与习题

9.1　简述汽车 CAN 总线方案是如何解决总线负荷过大问题的。

9.2　简述列车运行控制系统的三部分功能。

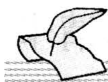
9.3　简述磁浮列车车载控制与诊断系统的结构组成。

参 考 文 献

[1] 邬宽明．CAN 总线原理和应用系统设计[M]．北京：北京航空航天大学出版社，1996．
[2] 阳宪惠．现场总线技术及其应用[M]．北京：清华大学出版社，2008．
[3] 饶运涛，邹继军，王进宏，郑勇芸．现场总线 CAN 原理与应用技术[M]．北京：北京航空航天大学出版社，2007．
[4] 杨春杰，王曙光，亢红波. CAN 总线技术[M].北京：北京航空航天大学出版社，2010．
[5] 牛跃听，周立功，方舟. CAN 总线嵌入式开发—从入门到实践[M].北京：北京航空航天大学出版社，2012.
[6] 来清民. 手把手教你学 CAN 总线[M] .北京：北京航空航天大学出版社，2010．
[7] 李真花，崔健. CAN 总线轻松入门与实践[M].北京：北京航空航天大学出版社，2011．
[8] 罗峰，孙泽昌. 汽车 CAN 总线系统原理、设计与应用[M].北京：电子工业出版社，2010.
[9] 张培仁. CAN 总线设计及分布式控制[M]. 北京：清华大学出版社，2012.
[10] 史久根，张培仁，陈真勇.CAN 现场总线系统设计技术[M]. 北京：国防工业出版社，2004.
[11] 王黎明，等．CAN 现场总线系统的设计与应用[M]．北京：电子工业出版社，2008．
[12] Holger Zeltwanger．现场总线 CANopen 设计与应用[M]．周立功，黄晓清，严寒亮，译．北京：北京航空航天大学出版社，2011．
[13] 周立功，等．iCAN 现场总线原理与应用[M]．北京：北京航空航天大学出版社，2007．
[14] 刘泽祥．现场总线技术[M]．北京：机械工业出版社，2005．
[15] 甘永梅，等．现场总线技术及其应用[M]．2 版．北京：机械工业出版社，2008．
[16] 王永华，等．现场总线技术及应用教程[M]．北京：机械工业出版社，2006．
[17] 邬宽明．现场总线技术应用选编 2[M]．北京：北京航空航天大学出版社，2004．
[18] 龙志强，李迅，刘建斌．计算机控制及网络技术[M]．北京：中国水利水电出版社，2007．
[19] 龙志强，李迅，李晓龙. 现场总线控制网络技术[M]. 北京：机械工业出版社，2011．
[20] Philips Semiconductors, CAN Specification Version 2.0, Parts A and B, 1992.
[21] Philips Semiconductors, SJA1000 Stand-alone CAN Controller，1999．
[22] Philips Semiconductors, APPLICATION NOTE SJA1000 Stand-alone CAN controller，1997．
[23] Philips Semiconductors, Data Sheet PCA82C250, CAN controller interface, 2000．
[24] Philips Semiconductors, Data Sheet PCA82C251, CAN controller interface, 2000．
[25] Philips Semiconductors,Data Sheet TJA1050, High speed CAN transceiver, 2000．
[26] Philips Semiconductors, Preliminary Data Sheet TJA1040, High speed CAN transceiver, 2001．
[27] Philips Semiconductors , Preliminary Data Sheet TJA1041, High speed CAN transceiver, 2001．
[28] 广州周立功单片机发展有限公司，PCA82C250/251 TJA1040 TJA1050 升级记录，译自 Philips 技术报告(HAI/TR0126), 2002．
[29] Dietmayer K. CAN Bit Timing, Philips Semiconductors, Technical Report HAI/TR9708, 1997．
[30] Dietmayer K, Overberg K W. CAN Bit Timing Requirements, SAE Paper 970295, 1997.

[31] Eisele-H, Jöhnk-E Application Note PCA82C250/251 CAN Transceiver, AN96116, Philips Semiconductors, 1996.
[32] Egon Jŏ hnk. Determination of Bit Timing Parameters for the CAN Control SJA1000，PHILIPS , 1997.
[33] SILICON LABS. C8051F50x-F51x Mixed Signal ISP Flash MCU Family data sheet, 2009.
[34] Texas Instruments Incorporated. TMS320C28x 系列 DSP 的 CPU 与外设[M]. 张卫宁，译. 北京：清华大学出版社，2004.
[35] TEXAS INSTRUMENGTS.TMS320F28335, TMS320F28334, TMS320F28332 Digital Signal Controllers（DSCs）Data Manual, 2007.
[36] TEXAS INSTRUMENGTS.TMS320x28xx,28xxx DSP Enhanced Controlled Area Network（eCAN）Reference Guide, 2002.
[37] 叶俊，姚煊道，龙志强. 基于 DeviceNet 显示报文传输协议的 CAN 网研究[C]. 2005 中国控制会议，2005.
[38] 朱双华. 汽车 CAN 系统故障诊断与检测技术[M]. 长沙：国防科技大学出版社，2008.
[39] 杨宏伟. 斯柯达明锐车 CAN 数据总线（一）[J]. 汽车维护与修理, 2011, 80-82.
[40] 李进，朱敏，宋健.基于 CAN 总线嵌入式工程机械控制系统研究[J]. 经济研究导刊，2012(4)：336-337.
[41] 唐修俊，李胜华，王巍. 基于 CAN 总线的汽车起重机智能控制系统[J]. 中国工程机械学报，2008，6(2)：990-992.
[42] 王剑波. 基于 CAN 总线的数字式自动找平控制系统设计[J]. 国外电子元器件，2007(2).
[43] 王剑波. 基于 CAN 总线的混凝土摊铺机自动找平控制系统研究[D]. 长沙：国防科技大学学位论文，2006.
[44] 陈子龙, 胡汉春. CAN 总线在地铁车辆中的应用[J]. 黑龙江科技信息，2011(34).
[45] 邓嘉, 赵志宏. 基于 CAN 总线的地铁车载 ATC 通信设计[J]. 南京工业职业技术学院学报，2010，10(2)：12-14.
[46] 郑万昀，国产化 ATP 系统中通信单元的设计与实现[J]. 铁路通信信号工程技术，2007，4(6)：41-42.
[47] 邵志和. LKJ2000 型列车运行监控记录装置的研究[D]. 长沙：中南大学学位论文，2005.
[48] 陈杨，刘曙生，龙志强. 基于 CAN 总线的数据通信系统研究[J]. 测控技术，2000，19(10)：53-55.
[49] 陈杨，龙志强. 基于 ISA 总线的 CAN 通讯卡的设计与实现[J]. 国防科技大学学报，2001，23(3)：115-119.
[50] 龙志强，任永平，刘曙生，陈杨. 磁浮列车车载计算机监控系统的可靠性设计[J]. 测控技术，2000，19(12)：37-40.
[51] 龙志强，刘曙生，佘龙华，尹力明，常文森. CMS-3 型磁悬浮列车微机监控与数据通信系统[J]. 机车电传动，2002(6)：21-24.
[52] 吕志国，吴军，施晓红，龙志强. 磁悬浮列车监控网络的编组自适应研究[J]. 测控技术，2003，22(9)：36-41.
[53] 龙志强，刘曙生，曹承侃. 基于 CAN 总线的磁悬浮列车通信网络研究[J]. 电气传动，2002(2)：44-47.
[54] 龙志强，黄晚德. 现场总线在香烟包装机生产线上的应用[J]. 工业仪表与自动化装置，2001(3)：42-48.
[55] 叶俊，刘曙生，龙志强. 一种基于 CAN 总线的 DSP 程序加载技术[J]. 电子技术应用，2003，29(11)：32-34.
[56] 洪华杰，龙志强. 基于 PLC 的磁悬浮列车信号监测系统研究[J]. 电子技术应用，2001，28(7)：39-42.
[57] 洪华杰，龙志强，钟慧娟，熊万龙. 磁悬浮列车 PLC 监测系统的通讯技术研究[J]. 电气传动，2002，32(6)：43-46.
[58] 钟慧娟，龙志强，洪华杰. 磁悬浮列车 PLC 故障自诊断方法研究[J]. 电气传动，2003，33(1)：40-50.

[59] 吴军，李晓龙，龙志强．CAN 控制网络实时性能分析与测量[J]．工业控制计算机，2004，17(10)：21-23．
[60] 刘志，龙志强．基于 OPC 的工业 PC 与 S7-300 通信的 VC 实现[J]．工业控制计算机，2008，21(5)：37-38．
[61] 龙志强，李晓龙，徐飞，辛华．“现场总线控制网络系统”课程的实验教学系统设计[J]．实验室研究与探索，2007，26(10)：44-47．
[62] 龙志强，刘曙生，陈扬．磁悬浮列车运行信号监测系统[J]．电子技术应用，2000，26(10)：25-30．
[63] 施晓红，佘龙华，龙志强．PC 与 PLC 的动态数据交换（DDE）研究[J]．电气传动，2005，35(4)：45-47．
[64] 吴军，龙志强．基于工业以太网和 CAN 总线的磁浮列车控制通信网络[C]．2004 全国空气动力测控技术交流会，2004．
[65] 吴军，龙志强．基于 DSP 嵌入式网关的磁浮列车通信网络[C]．中国航空学会信号与信号处理专业全国第八届学术会议，2004．
[66] Zhiqiang Long, Guang He, Song Xue. Study of EDS&EMS Hybrid Suspension System With Permanent-Magnet Halbach Array[J]. IEEE TRANSACTION ON MAGNETICS，2011，47(12)：4717-4724.
[67] Zhiqiang Long, Guang He, Ning He, Zhizhou Zhang. Experimental Research on an Energy-saving Maglev Train[J]. Environmental Engineering and Management Journal，2011，10(7)：975-979.
[68] 龙志强，蔡楹，徐听. 基于分布式计算法的磁浮列车故障综合评判[J]. 控制与决策，2009，24(4)：551-556.
[69] 李云，龙志强. 磁悬浮系统网络化悬浮控制器的设计与实现[J]. 系统仿真学报，2009，21(14)：4420-4424.
[70] 周纯杰，于跃，徐邦荃. 基于 CAN 协议的新型染整设备现场总线控制系统[J]. 电气传动，2002(1)：40-43.
[71] 周纯杰，刘华成，尤兵，陶志东，周孝惠，文伟. 文本型字符显示器中可重构通信的方法与实现[J]. 计算机工程与科学，2010(2)：142-145.
[72] 湖北黄石科威自控有限公司，嵌入式 PLC EASY 编程手册.
[73] 湖北黄石科威自控有限公司，嵌入式 PLC EASY 原理及应用.
[74] 湖北黄石科威自控有限公司，混合型可编程逻辑控制器使用说明书.
[75] 湖北黄石科威自控有限公司，AD1216 使用说明书.
[76] 秦石桥，王省书，黄勇. 微机接口技术及应用[M]. 长沙：国防科技大学出版社，2000.



本科电气精品教材推荐

电气信息工程丛书

西门子工业通信网络组态编程与故障诊断

书号： 28256　**定价：** 69.00 元

作者： 廖常初　**配套资源：** DVD 光盘

推荐简言：

本书建立在大量实验的基础上，详细介绍了实现通信最关键的组态和编程方法，随书光盘有上百个通信例程，绝大多数例程经过硬件实验的验证。读者根据正文介绍的通信系统的组态步骤和方法，参考光盘中的例程作组态和编程练习，可以较快地掌握网络通信的实现方法。

西门子 PLC 高级应用实例精解

书号： 29304　**定价：** 42.00 元

作者： 向晓汉　**配套资源：** DVD 光盘

推荐简言：

本书通过实例全面讲解西门子 S7-200/S7-1200/S7-300 PLC 的高级应用。内容包括梯形图的编程方法、PLC 在过程控制中应用、PLC 在运动控制中的应用、PLC 的通信及其通信模块的应用等。书中实例都用工程实际的开发过程详细介绍，便于读者模仿学习。每个实例都有详细的软件、硬件配置清单，并配有接线图和程序。本书所附配套资源中有重点实例源程序和操作过程视频文件。

西门子 WinCC V7 基础与应用

书号： 32902　**定价：** 44.00 元

作者： 甄立东　**配套资源：** DVD 光盘

推荐简言： 本书系统地介绍了 WinCC V7.0 的功能及其组态方法。首先介绍了初级用户必须掌握的主要功能，其次介绍了高级用户需要了解的 Microsoft SQL Server 2005、冗余系统组态、全集成自动化、开发性和工厂智能选件。通过实例，详尽地展示了各种应用的设计和实现步骤以及应用。本书还对 WinCC V7.0 新增功能进行了详细讲解。

现场总线与工业以太网及其应用技术

书号： 35607　**定价：** 58.00 元

作者： 李正军　**配套资源：** 电子教案

推荐简言： 本书从工程实际应用出发，全面系统地介绍了现场总线与工业以太网技术及其应用系统设计，力求所讲内容具有较强的可移植性、先进性、系统性、应用性、资料开放性，起到举一反三的作用。主要内容包括现场总线与工业以太网概论、控制网络技术、通用串行通信接口技术、PROFIBUS 现场总线、PROFIBUS-DP 通信控制器与网络接口卡等。

PLC 编程及应用 第 3 版

书号： 10877　**定价：** 37.00 元

作者： 廖常初　**配套资源：** DVD 光盘

获奖情况： 全国优秀畅销书

推荐简言： 西门子公司重点推荐图书，累计销量已达 12 万册。本书以西门子公司的 **S7-200 PLC** 为例，介绍了 **PLC** 的工作原理、硬件结构、指令系统、最新版编程软件和仿真软件的使用方法；介绍了数字量控制梯形图的一整套先进完整的设计方法；介绍了 **S7-200** 的通信网络、通信功能和通信程序的设计方法等。配套光盘有 **S7-200** 编程软件和 **OPC** 服务器软件 **PC Access**、与 **S7-200** 有关的中英文用户手册和资料、应用例程等

S7-300/400 PLC 应用技术 第 3 版

书号： 36379　**定价：** 69.00 元

作者： 廖常初　**配套资源：** DVD 光盘

推荐简言： 西门子公司重点推荐图书，销量已达 8 万册。本书介绍了 S7-300/400 的硬件结构、性能指标和硬件组态的方法；指令系统、程序结构、编程软件 STEP7 的使用方法；梯形图的经验设计法、继电器电路转换法和顺序控制设计法，以及使用顺序功能图语言 S7 Graph 的设计方法。另外还介绍了 S7-300/400 的网络结构，AS-i 和工业以太网、PRODAVE 通信软件的组态、参数设置的编程的方法。配套的光盘附有大量的中英文用户手册、软件和例程，附有 STEP7 编程软件。

机工出版社·计算机分社书友会邀请卡

尊敬的读者朋友：

感谢您选择我们出版的图书！我们愿以书为媒与您做朋友！我们诚挚地邀请您加入：

“机工出版社·计算机分社书友会”

以书结缘，以书会友

加入“书友会”，您将：

★ 第一时间获知新书信息、了解作者动态；

★ 与书友们在线品书评书，谈天说地；

★ 受邀参与我社组织的各种沙龙活动，会员联谊；

★ 受邀参与我社作者和合作伙伴组织的各种技术培训和讲座；

★ 获得“书友达人”资格（积极参与互动交流活动的书友），参与每月 5 个名额的“书友试读赠阅”活动，获得最新出版精品图书 1 本。

如何加入“机工出版社·计算机分社书友会”

两步操作轻松加入书友会

Step1

访问以下任一网址：

★ 新浪官方微博：http://weibo.com/cmpjsj

★ 新浪官方博客：http://blog.sina.com.cn/cmpbookjsj

★ 腾讯官方微博：http://t.qq.com/jigongchubanshe

★ 腾讯官方博客：http://2399929378.qzone.qq.com

Step2

找到并点击调查问卷链接地址（通常位于置顶位置或公告栏），完整填写调查问卷即可。

联系方式

通信地址：北京市西城区百万庄大街 22 号
机械工业出版社计算机分社
邮政编码：100037

联系电话：010-88379750
传　　真：010-88379736
电子邮件：cmp_itbook@163.com

敬请关注我社官方微博： http://weibo.com/cmpjsj

第一时间了解新书动态，获知书友会活动信息，与读者、作者、编辑们互动交流！

检 12

在线互动交流平台

官方微博：http://weibo.com/cmpjsj

豆 瓣 网：http://site.douban.com/139085/

读者信箱：cmp_itbook@163.com

电气信息工程丛书

- ◇ PLC 编程及应用（第 3 版）
- ◇ S7-300/400 PLC 应用技术（第 3 版）
- ◇ 西门子 S7-200 PLC 编程及应用案例精选
- ◇ 西门子人机界面(触摸屏)组态与应用技术
- ◇ 西门子工业通信网络组态编程与故障诊断
- ◇ 西门子 WinCC V7 基础与应用
- ◇ 三菱 PLC 工程应用设计
- ◇ CS/CJ 系列 PLC 应用基础及案例
- ◇ 欧姆龙 CP1H PLC 应用基础与编程实践
- ◇ IEC 61131-3 编程语言及应用基础
- ◇ 网际组态软件 Advantech WebAccess 应用技术
- ◇ 组态软件基础与工程应用(易控 INSPEC)
- ◇ Protel 99 SE 原理图与 PCB 设计及电路仿真
- ◇ Protel 99 SE 实战详解与技巧
- ◇ Protel 2004 电路原理图及 PCB 设计
- ◇ 基于 Altium Designer 的原理图与 PCB 设计
- ◇ 基于 Allegro 的 PCB 设计与开发
- ◇ VHDL 数字电路及系统设计
- ◇ FPGA/VHDL 设计入门与进阶
- ◇ OrCAD 电路原理图设计与应用
- ◇ 单片机应用及 51 程序设计（第 2 版）
- ◇ 51 单片机快速上手
- ◇ 数字信号处理器原理、结构及应用基础——TMS320F28x
- ◇ DSP 技术及应用系统设计
- ◇ Freescale 9S12 十六位单片机原理及嵌入式开发技术
- ◇ 基于 EDK 的 FPGA 嵌入式系统开发
- ◇ 基于模型的设计——Qsys 篇
- ◇ 嵌入式可配置实时操作系统eCos开发与应用（第 2 版）
- ◇ Linux PowerPC 详解——核心篇
- ◇ 典型数控系统应用技术（FANUC 篇）
- ◇ 现场总线控制网络技术
- ◇ 现场总线与工业以太网及其应用技术
- ◇ 数控原理与编程
- ◇ 风电并网运行与维护
- ◇ 无线传感器网络及其在物流中的应用
- ◆ CAN 总线技术与应用系统设计

地址：北京市百万庄大街22号
邮政编码：100037
电话服务
社服务中心：010-88361066
销售一部：010-68326294
销售二部：010-88379649
读者购书热线：010-88379203
网络服务
教材网：http://www.cmpedu.com
机工官网：http://www.cmpbook.com
机工官博：http://weibo.com/cmp1952
封面无防伪标均为盗版

上架指导 工业技术/总线技术

ISBN 978-7-111-41867-2

策划编辑◎时静 / 封面设计◎刘吉维



定价：39.90 元